# This Page Is Inserted by IFW Operations and is not a part of the Official Record

# BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

# IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents will not correct images, please do not report the images to the Image Problem Mailbox.

THIS PAGE BLANK (USPTO)



# United States Patent [19]

Theimer et al.

[11] Patent Number:

5,649,099

[45] Date of Patent:

Jul. 15, 1997

[54] METHOD FOR DELEGATING ACCESS
RIGHTS THROUGH EXECUTABLE ACCESS
CONTROL PROGRAM WITHOUT
DELEGATING ACCESS RIGHTS NOT IN A
SPECIFICATION TO ANY INTERMEDIARY
NOR COMPRISING SERVER SECURITY

[75] Inventors: Marvin M. Theimer; David A. Nichols, both of Mountain View; Douglas B. Terry, San Carlos, all of Calif.

[73] Assignee: Xerox Corporation, Stamford, Conn.

[21] Appl. No.: 71,649

[22] Filed: Jun. 4, 1993

[56] References Cited

U.S. PATENT DOCUMENTS

#### OTHER PUBLICATIONS

Eichin and Rochlis, "With Microscope and Tweezers: An Analysis of the Internet Virus of Nov. 1988", 1989, pp. 326-343.

- J. E. Donnelley et al. "Resource Access Control in a Network Operating System," Univ. of CA, Lawrence Livermore Lab. pp. 115-125.
- R. M. Needham et al. "Using Encryption for Authentication in Large Networks of Computers," Communications of the ACM, Dec. 1978, vol. 21, No. 12 pp. 993-999.
- D. E. R. Denning "Cryptography and Data Security," (execrpts) '82, pp. 7-16, 56-58, 101-126, 129-133, 161-179, 187-190, 191-329.

Russel Sandberg et al. "Design and Implementation of the Sun Network Filesystem," USENIX Association Summer Conference Proc. Portland, 1985, Jun. 11-14,1985, pp. 119-130.

A. S. Tanenbaum "Using Sparse Capabilities in a Distributed Operating System," The 6th Intl. Conf. on Distributed Computing Systems, Cambridge, Mass., May 19-23, 1986, pp. 558-563.

Richard Stallman "GNU Emacs Manual," Fifth Edition, Emacs Ver. 18, Oct. 1986.

Xerox, "Authentication Protocol," Xerox System Integration Standard, May 1986.

James Gosling "SUNDEW: A Distributed and Extensible Window Sys.," USENIX, '86 Winter USNIX TECH. CONF., Colorado pp. 98-103.

J. G. Steiner "Kerberos: An Authentication Service for Open Network Syst." USENIX Winter Conf. Feb. 9-12, '88, TX, pp. 191-202.

B.N. Bershad "Watchdogs: Extending the UNIX File System," USENIX Winter Conference Feb. 9-12, 1988, Texas pp. 267-275.

J.W. Stamos et al. "Remote Evaluation," ACM Trans. on Programming Languages and Systems, vol. 12, No. 4, Oct. 1990 pp. 537-565.

K. R. Sollins "Cascaded Authentication," Proceedings 1988 IEEE Symposium on Security and Privacy Apr. 18–21, '88, CA pp. 156–163.

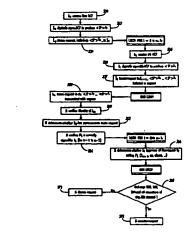
(List continued on next page.)

Primary Examiner—Lance Leonard Barry, Esq. Assistant Examiner—Le Hien Luu Attorney, Agent, or Firm—Alexander E. Silverman

7] ABSTRACT

A method in which access control programs (ACPs) permit controlled delegation of access rights from clients to untrusted intermediaries. ACPs are programs that encode arbitrary specifications of delegated access rights. In the method, a client creates an ACP and associates it with a request to a server, the request being made through one or more intermediaries. When processing a request received from an intermediary, the server executes the access control program to determine whether or not to grant the request.

36 Claims, 10 Drawing Sheets



#### OTHER PUBLICATIONS

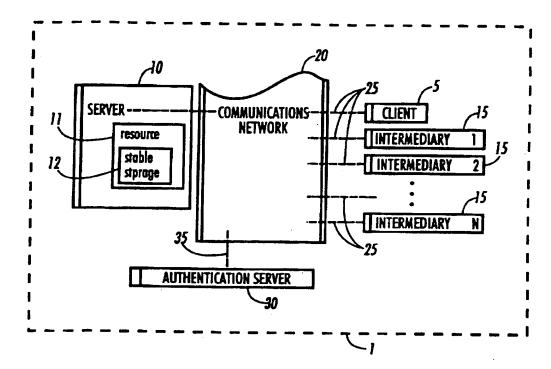
J. H. Howard et al. "Scale and Performance in a Distributed File System," ACM Transactions on Computer Systems, vol. 6, No. 1, Feb. 1988, pp. 51-81.

M. Satyanarayanan "Integrating Security in a Large Distributed System," ACM Transactions on Computer Systems, vol. 7, No. 3, Aug. 1989, pp. 247-280.

Morrie Gasser et al. "An Architecture for Practical Delegation in a Distributed System," CH2884-5/90/0000/0020\$01.00 © 1990 IEEE pp. 20-30.

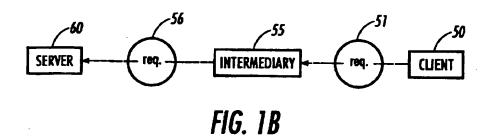
J. K. Ousterhout "Tel: An Embeddable Command Language," USENIX Association of the Winter 1990 USENIX Conference, Jan. 22-26, 1990, Washington, D.C., USA pp. 133-146.

SunOS™ Reference Manual, Sun Microsystems, (excerpts) Comm. Commands, Misc. Ref. Man. Pages, Chapt. 1, 4, 8 and 14.



Jul. 15, 1997

FIG. 1A



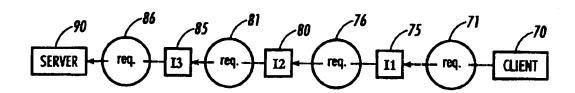


FIG. 1C

Jul. 15, 1997

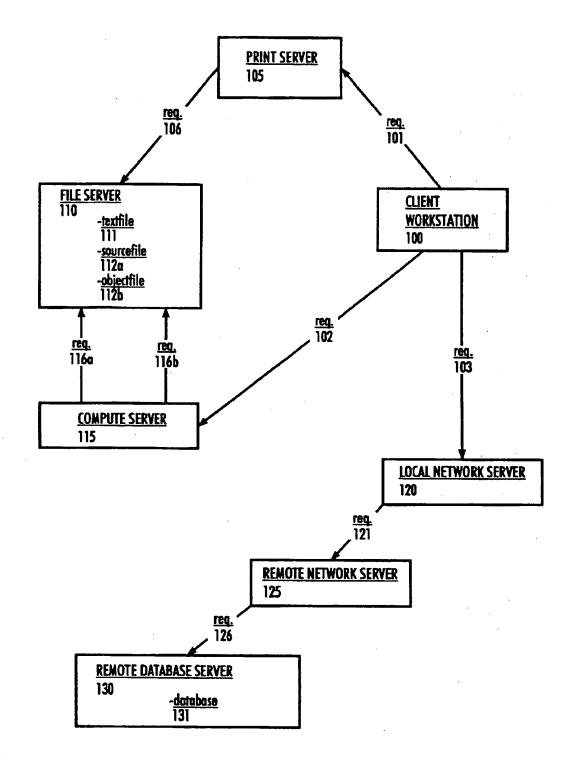


FIG.1D

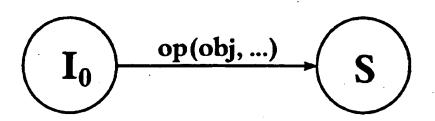


FIG.2A (Prior Art)

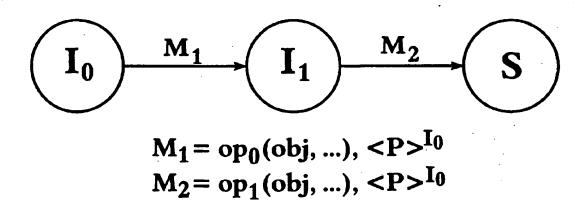


FIG.2B

Jul. 15, 1997

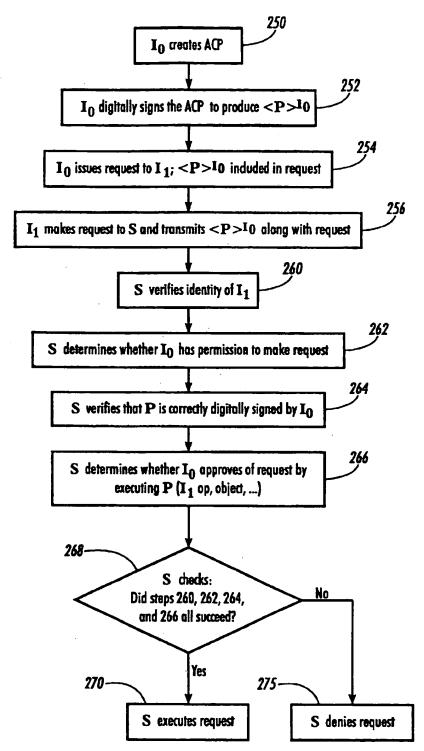
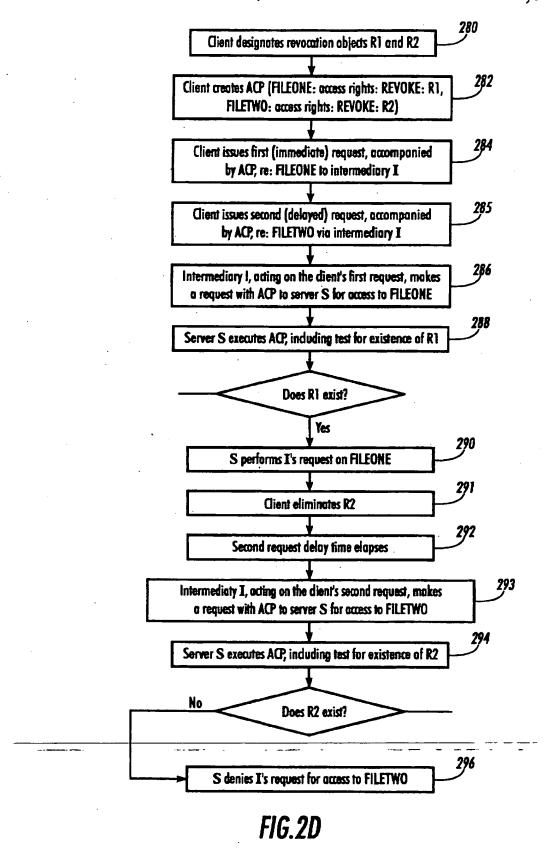
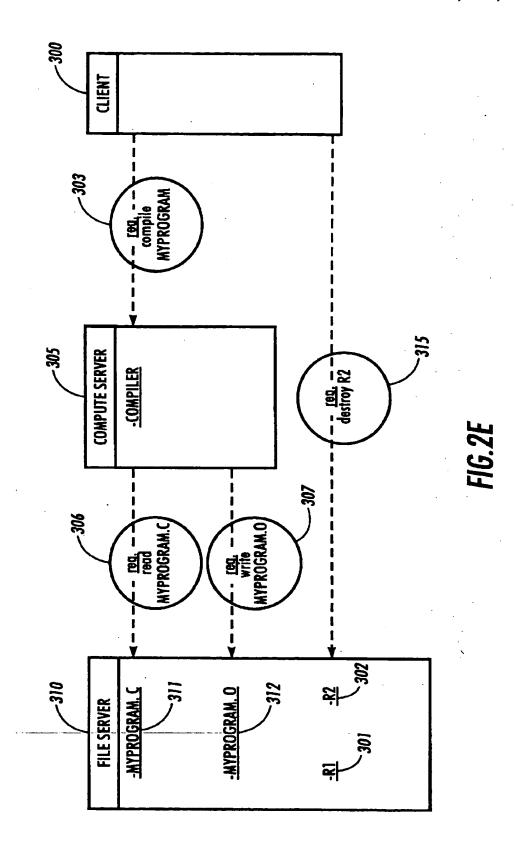


FIG.2C





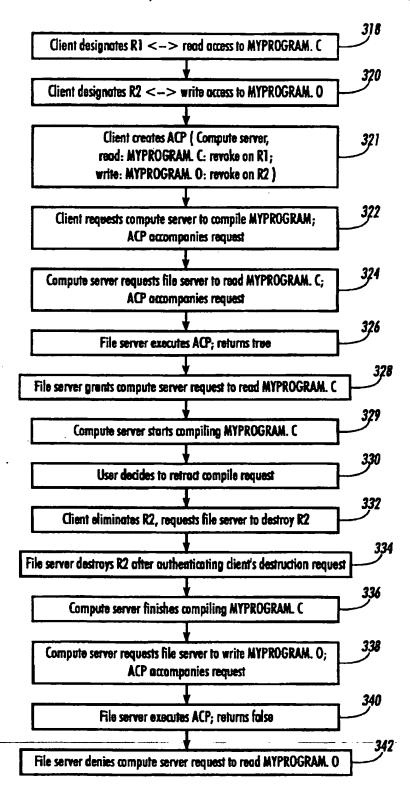
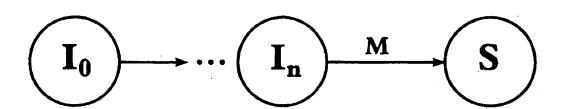
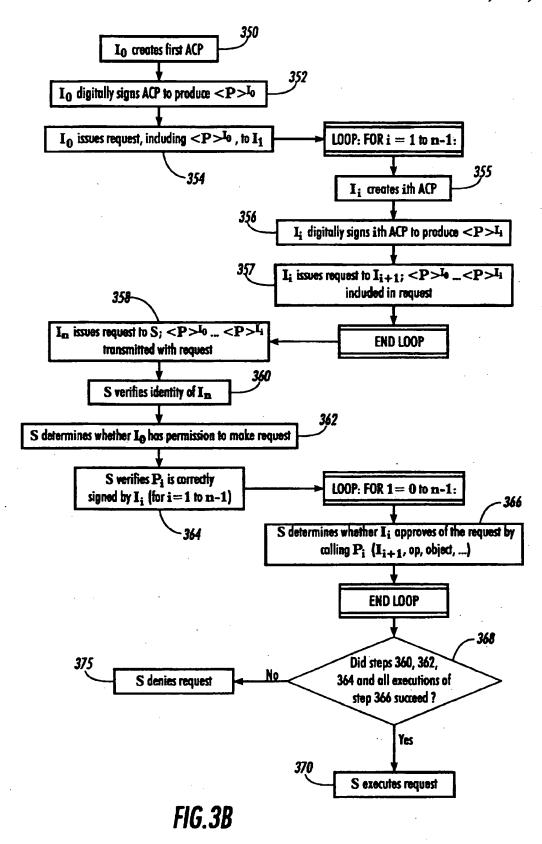


FIG.2F



 $\mathbf{M} = \mathrm{op(obj, ...)}, <\!\! \mathbf{P_0} \!\!>^{\!\! \mathbf{I}} \!\! \mathbf{0}, <\!\! \mathbf{P_1} \!\!>^{\!\! \mathbf{I}} \!\! \mathbf{1}, ... <\!\! \mathbf{P_{n-1}} \!\!>^{\!\! \mathbf{I}} \!\! \mathbf{n}_{\!\! -1}$ 

FIG.3A



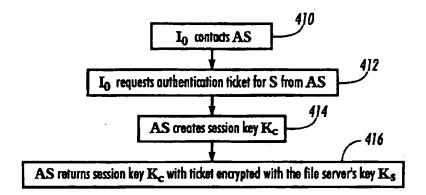


FIG.4A (Prior Art)

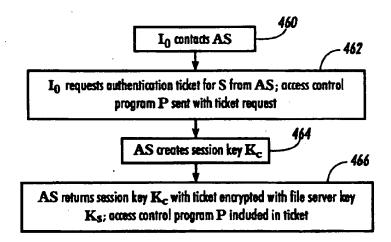


FIG.4B

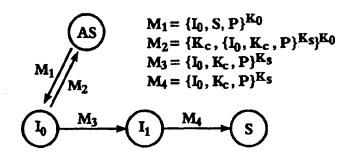


FIG.4C

METHOD FOR DELEGATING ACCESS RIGHTS THROUGH EXECUTABLE ACCESS CONTROL PROGRAM WITHOUT DELEGATING ACCESS RIGHTS NOT IN A SPECIFICATION TO ANY INTERMEDIARY NOR COMPRISING SERVER SECURITY

#### COPYRIGHT NOTIFICATION

A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owners have no objection to the facsimile reproduction, by anyone, of the patent document or the patent disclosure, as it appears in the patent and trademark office patent file or records, but otherwise reserve all copyright rights whatsoever.

#### Software Appendix

An appendix comprising two printed files is included as part of this application. The first file is entitled "Diffs," and 20 is 28 pages long plus a cover sheet. It represents the differences in source code between release 3 of the Andrew File System (as it appeared on the Aug. 19, 1990 Mt. Xinu release tape of the Mach operating system) and a modified version of release 3 of the Andrew File System, described 25 below, that embodies the method of the present invention. The second file is entitled "TCLDiffs" and is 25 pages long plus a cover sheet. It represents the differences in source code between release 3.3 of Tool Command Language and a version of Tool Command Language modified from release 30 3.3 to support the embodiment of the invention in the modified version of the Andrew File System as described below. The modifications made to Andrew File System release 3 and Tool Command Language release 3.3 in order to produce the embodiment of the present invention in the 35 modified version of the Andrew File System as described below represent unpublished work, Copyright @ 1991 Xerox Corporation. All rights reserved. Copyright protection claimed includes all forms and matters of copyrightable material and information now allowed by statutory or judi- 40 cial law or hereafter granted, including without limitation, material generated from the software programs which are displayed on the screen such as icons, screen display looks,

#### BACKGROUND OF THE INVENTION

The present invention relates to computing systems and more particularly to client-server systems, including but not limited to distributed client-server systems.

In a distributed client-server system, a client program often requires an intermediary to perform an operation on some server. The intermediary must be able to convince the server that it is operating on behalf of the client and hence that it should be granted the right to perform the requested operation. Furthermore, to limit exposure to untrusted intermediaries, the client will want to grant to the intermediary only that subset of its rights that are necessary for completing the requested operation.

An example of such a situation is the use of a print server 60 to print a file that resides on a file server. The initiating user would like to grant the print server access to the file to be printed so that it can directly retrieve the file from the file server. However, the user would like to prevent the print server from being able to retrieve any other files. The user 65 might also wish to place a time limit on how long the print server has access rights to the file.

A second example is remote compilation on a compute server. The compute server must be given read access to all the relevant source files. It should also be permitted to create or overwrite the relevant object files while being prevented from modifying source files. Additionally, suppose the compute server does a recompilation as a set of parallel tasks, each running on a separate machine. Then the server will want to delegate some of the rights it has acquired to other hosts that are performing subtasks.

This example can be further complicated if the files and directories involved are owned by someone other than the user who is invoking the compilation. The user (e.g., as member of a group) may have permission to read and write various files in a directory, but may lack permission to modify the access controls of those files or the directory they are in. Hence it may not even be possible for the user to delegate access rights to a third party.

A more application-specific example is a self-paced course in which students submit their homework assignments by creating a file in a well-known directory. The solutions to each homework assignment also reside in files in that directory; however, each student should only be given access to the solutions after they have handed in their answers for that assignment. The access control specification in this case is that only students in the class may read or write files in the directory, no student in the class may read a solution file without having already written a corresponding homework file and no student may write any given homework file more than once.

Finally, one may wish to limit access to resources that can be subdivided, such as electronic funds. This implies being able to specify quantitative limits on resources for which access rights have been delegated.

Techniques proposed in the prior art for granting/
delegating access control have centered around access control lists and capability schemes. Access control lists (ACLs)
are lists of (name, access right) tuples. Such lists may be
implemented as bit tables, linked lists, or other suitable data
structures. Servers maintain ACLs and use them to decide
whether or not to grant any given access request. Capability
schemes are based on capability tokens that servers hand out
to clients. A requestor, such as an intermediary between a
client and a server, presents a capability token along with an
access request to prove that the requestor has the right to
make the request of the server.

Both ACL- and capability-based systems provide ways for a client to delegate its access rights to an intermediary, but provide only limited facilities for restricting the rights granted to the intermediary. ACL-based systems can deal with restricted delegation by allowing the creation of roles, which explicitly represent the entity to whom a restricted set of access rights is being delegated. Capability-based systems enable restricted delegation by either handing out multiple tokens or by handing out tokens that can be securely subsetted to a certain degree. Both of these approaches to restricted delegation depend on servers' having an explicit understanding of all access controls: Concepts such as restrictions over file types, access time limits, the homework example restrictions, or resource quotas must be implemented at the servers. Servers must know in advance of any client requests all the various access rights and restrictions that clients may want to delegate.

With either ACL- or capability-based systems, if a client wishes to enforce access controls that are not understood by the server(s) available to him, he has only one option, namely, to use or build other servers. For example, consider

a distributed file system that does not ordinarily support access time limits. An example is the Unix distributed file system known as NFS, which is described, for example, in Russell Sandberg, David Goldberg, Steve Kleiman, Dan Walsh, and Bob Lyon, "Design and Implementation of the Sun Network File System," in Proceedings of the Summer 1985 USENIX Conference (Portland, Ore., June 1985) at 119-130. To build a print service that understands access time limits on NFS files requires that a new print server be built that both understands time-limit-based access controls and is trusted with access to all NFS files that any clients might wish to print. Similarly, to implement the homework example on top of prior art file servers (or any file servers whose access controls lack the concepts necessary to express the homework constraints) requires that someone build a "homework server." Furthermore, students' ability to hand 15 their homework in and receive solution sets back depends on the availability of this homework server.

To better understand the limitations of the prior art, it is helpful to consider an analogy. Suppose that a movie theater shows ten different movies. Some are suitable for viewers of 20 all ages, while others are suitable only for adults. Like most movie theaters, this theater sells tickets separately for each individual movie. A patron who holds a ticket for, say, "Gone With the Wind," is thereby entitled to see "Gone With the Wind" but is not entitled to see "Jaws" or "Unforgiven" 25 or any other movie playing at the theater.

Now suppose that a parent wishes to send her child, who is thirteen years old and not to be trusted, to see a movie unaccompanied. The parent wants the child to see only "Grated" movies, that is, movies deemed suitable for view- 30 ers of all ages, and no other movies. She is concerned that if she simply gives the child money to purchase a ticket, left to his own devices the child (possibly with the assistance of an adult or older teenager posing as his "guardian") will purchase a ticket to an "R-rated" movie intended for more mature viewers. Thus the parent runs the risk that if she tells 35 her child to go see "Bambi," the child will sneak in to see "Basic Instinct." What the parent really wants is to be able to purchase in advance, and give to her child, a movie ticket redeemable for access to any G-rated movie and for no other movies. Unfortunately the movie theater does not sell such 40 tickets. Short of persuading the movie theater to change its ticket-selling policies, the parent is stuck with either having to accompany her child to the theater or else running the risk that the child will disobey her.

It can be seen that the parent is analogous to a client 45 program, the child to an intermediary, the movie theater to a server, and the movie ticket to a capabilities token in a capability-based system. The parent-client is stuck with the kind of tickets that the theater-server sells, and cannot order independence to her intermediary-child while simultaneously maintaining a certain measure of control over the intermediary-child's behavior. The system provides no straightforward way for a client to design a restricted set of access privileges, e.g., at run time, and delegate these to a 55 potentially untrustworthy intermediary.

What is needed as an alternative to embedding an everincreasing multitude of access control concepts into each server or building an ever-increasing set of applicationspecific "front-end" servers is to provide clients and servers 60 with a language with which they can dynamically build generalized capabilities and define application-specific access rights at the time those rights are to be delegated.

#### SUMMARY OF THE INVENTION

The present invention provides a method for delegation in client-server systems that is based on client-manufactured

access control programs (ACPs) that can encode arbitrarily general access rights specifications. According to one aspect of the invention, an intermediary making a request of a server on behalf of a client presents an appropriate clientcreated ACP along with the request. The server executes the ACP to determine whether or not the requestor—that is, the intermediary—has been granted by the client the right to make the given request. If and only if the requestor has been granted the right to make the given request does the server carry out the request. In another aspect of the invention, two or more intermediaries stand between the client and server. Each intermediary creates its own additional ACP in order to delegate some or all the rights it has to the next intermediary in the chain. The server checks all the ACPs before carrying out a request issued by the last intermediary in the chain.

An important advantage of the method of the present invention is that a wide range of applications and access control schemes can be built on top of any server that implements an ACP interpreter and a relatively small number of access control concepts. Another advantage is that the method finds applicability in distributed and nondistributed systems. Still another advantage is that ACPs can be digitally signed to prevent their forgery, thus allowing them to be freely passed around or even published. A further understanding of the nature and advantages of the present invention may be realized by reference to the remaining portions of the specification and the drawings.

#### BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1A schematically depicts a client-server system suitable to the method of the present invention;

FIG. 1B schematically depicts a client request and an intermediary request that is also a service request;

FIG. 1C schematically depicts a client request, a first intermediary request, and additional intermediary requests, the last of which is also a service request;

FIG. 1D schematically depicts an example of multiple contemporaneous requests;

FIG. 2A schematically depicts an authenticated remote procedure call with no delegation of rights, as in the prior

FIG. 2B schematically depicts a remote procedure call with delegation of rights via an access control program;

FIG. 2C is a flowchart illustrating the steps of a remote procedure call with delegation via an access control program;

FIG. 2D is a flowchart illustrating the steps for revoking a custom-made ticket that would allow her to grant some 50 access privileges using revocation objects with an example

> FIG. 2E schematically depicts an example that illustrates the use of revocation objects in a situation wherein a single client request generates two intermediary requests;

> FIG. 2F is a flowchart illustrating the sequence of steps followed in the example of FIG. 2E;

> FIG. 3A schematically depicts a remote procedure call with chained delegation:

FIG. 3B is a flowchart illustrating the steps of a remote procedure call with chained delegation;

FIG. 4A is a flowchart illustrating the steps of the authentication protocol in the Andrew File System of the prior art;

FIG. 4B is a flowchart illustrating the steps of the authen-65 tication protocol in a version of the Andrew File System modified according to a specific embodiment of the present invention: and

5

FIG. 4C schematically depicts an access control program passed in an authentication ticket via a remote invocation facility.

#### DESCRIPTION OF SPECIFIC EMBODIMENTS

The disclosures in this application of all articles and references, including patent documents, are incorporated herein by reference. Also incorporated by reference are sections 1C, 2V, and 3V of the SUN Solaris 1.1 User Manual, respectively for their descriptions of the Unix command rsh, the Unix command stat, and the Unix setuid facility.

#### 1. Introduction

#### 1.1 Overview and Design Considerations

To get an overview of ACPs, it is helpful to begin with some illustrative analogies. Returning for a moment to the movie theater analogy given earlier, recall that the parent 20 wants to be able to give her child access to any G-rated movie but to no other movies. The parent would like to be able to purchase an advance ticket redeemable for G-rated movies. Better still, the parent would like to avoid the trouble of advance purchase. She would like to be able to write a check for the price of admission to the theater, say \$7, with special instructions on the check as follows: "Sell my Child a ticket to a G-rated movie only; pay to Theater \$7 for this purpose. Signed, Parent." The theater would accept this check in lieu of cash and give the child access to any G-rated film. Of course, the theater would first ask the child for identification, to be sure that the child was indeed the Child mentioned on the check. Also the theater would check Parent's signature to ensure that the check was not forged.

Translating this example to the client-server domain, the signed check with special instructions is analogous to an ACP of the present invention. The parent-client writes a "check"—an ACP—that grants the child-intermediary only those access rights that the parent-client specifically delegates. The child-intermediary presents the check to the theater-server, who confirms the child-intermediary's identity, verifies the parent-client's digital signature on the ACP, and then executes the ACP to determine whether to grant the child-intermediary access to a particular movie-resource.

In another analogy, consider a person who signs a powerof-attorney document. The power-of-attorney document delegates to an attorney certain rights to do on the person's behalf what the person could do for himself, for example, to draw funds from the person's bank account. Importantly, the 50 person can delegate to the attorney only such rights as he actually has; he cannot, for example, give the attorney the right to draw funds from somebody else's bank account. The person signs the power-of-attorney document to prove that the document is genuine. When the attorney presents the 55 document to a third party, such as a bank, the attorney also presents her own identification to show that the attorney is who she purports to be. It can be seen that the person is analogous to a client, the attorney to an intermediary, the bank to a server, the client's bank account to a resource 60 controlled by the server, and the power-of-attorney document to an ACP. Before allowing the attorney-intermediary to draw funds, the bank-server confirms the attorneyintermediary's identity and authenticates the ACP by verifying the person-client's digital signature attached to the 65 ACP. Thereafter the bank-server reads the power-of-attorney document-that is, executes the ACP. If all is in order, the

bank-server then allows the attorney-intermediary to draw funds from the person-client's account-resource on the person-client's behalf.

Several considerations arise in the design of an architecture incorporating access control programs. These include: how ACPs are transmitted between the interested parties-that is, between the client, server, and intermediarles; how ACPs are used by the servers that receive them; how powerful the language that is used to write ACPs should be; 10 how ACPs are revoked once issued; and how ACPs can be used to restrict the rights delegated by other ACPs. The following sections of the description address these considerations and also describe a specific embodiment of the invention that integrates ACPs into a particular file system 15 called the Andrew File System.

#### 1.2 Glossary

The following terms are intended to have the following general meanings:

AFS: "Andrew File System": a particular distributed file system.

Access Right: The right to obtain access to or use some resource. Examples: the right to obtain read access for a particular file; the right to execute a particular program. Synonyms: "right"; "privilege."

Access Control List: A list of access rights for a particular resource.

Access Control Program: A programmatic specification of access rights.

Authentication: The act of checking that something or someone is what they claim they are.

Authentication Server: A trusted program or machine that can authenticate various things and/or that can issue encrypted data items that others can use for authentication purposes.

Authentication Ticket: The encrypted data items that an authentication server provides to its clients for authentication purposes.

Capability: A token of some sort that can be used as proof of some access right.

Check (name, resource, request): A function used by a server in some embodiments of the present invention. This function checks whether the user or process whose name is "name" has access rights to "resource" that allow "request" to be performed on "resource".

Client: A program that desires access to some resource or operation controlled by a server.

Client Request: A request for access to some resource or operation that is controlled by a server.

Client-Server System: Any system in which resources are managed/controlled by server programs/processes and requests for access to and use of those resources are made by client programs/processes.

Communications Channel: A means by which two parties can reliably communicate with each other.

Create (resourceName, revocationName): A function that a client can request of a server that will create a revocation object with name "revocationName" for the server's resource named by "resourceName".

Destroy (revocationName): A function that a client can request of a server that will destroy an existing revocation object whose name is "revocationName". Only the client who created a revocation object may destroy it.

Digital Signature: A property private to a user or process that is used for signing sequences of data items. A digital

signature for a sequence of data items is a unique data value that is a function of the private property and the sequence of data items to be signed.

Initiator: The original client of a request; the one who has access rights allowing the request without need for delegation of access rights from some other party.

Intermediary: Someone making a request on behalf of some other client. An intermediary must receive delegated access rights in order to successfully make a request to a

Intermediary Request: The request made by an intermediary on behalf of some other client.

Kerberos protocol: A particular authentication protocol.

Key: As in encryption key.

NFS: "Network File System"-a particular distributed file system.

Privilege: See "access right."

Revocation Object: An object controlled by a server that can be destroyed at the request of a client in order to invalidate all or part of an access control program that that client has created.

Right: See "access right."

Role: An artificial user identity that has been created in 25 order to specify a specific set of access rights; for example, those pertaining to some particular job or administrative

rsh: A Unix command that executes a program on another machine from the one that "rsh" is invoked on.

Secure Channel: A communications channel whose contents cannot be monitored by third parties.

Server: A program that controls resources that various clients of a system may wish to use or gain access to.

Service Request: A request to a server for access to or use of some resource.

stat: A Unix system call that returns information about a particular file, including its last modification time and who may access it in what fashion.

Template: A prototype for an access control program, containing various fields that must be filled in with specific values in order to instantiate a specific access control program.

Thread of Computation: A specific sequence of computer instructions actually executed by a computer. A computer can interleave the execution of multiple threads of computation by running multiple programs "simultaneously".

#### 2. The Method

## 2.1 System configuration

FIG. 1A illustrates a system configuration suitable to the 5, server 10, and one or more intermediaries 15, all connected to one another via communications network 20 that comprises a plurality of channels 25. System 1 can in some embodiments further comprise additional servers, clients, and intermediaries (not shown). Server 10 manages a resource 11 to which client 5 has access rights. Client 5 wants to delegate some or all of these rights to one or more of the intermediaries 15.

Client 5, server 10, and intermediaries 15 are processes that execute on one or more computing nodes (not shown). 65 Each process is a software entity that comprises an address space (or protection domain), one or more threads of

execution, and possibly one or more input/output streams. Each node is a computer comprising memory and one or more processors. Typically, but not always, processes are associated one-to-one with nodes, so that each process executes on its own node. Threads need not be associated one-to-one with processors.

It is assumed that each process is protected from every other process. Thus where all the processes-client, server, and intermediaries-execute on a single node, each process 10 has its own protected address space in the node's memory. Where different processes execute on different nodes, each node is assumed to have its own individual memory that cannot freely be accessed by other nodes.

The processes communicate with one another through communications network 20. Communications network 20 comprises channels 25 through which the various processes can communicate with one another. Where all the processes execute on a single node, the communications network is a set of interprocess communication links that also execute on the node. Where different processes execute on different nodes, the communications network comprises hardware links and software protocols through which the nodes can communicate with one another. The communications network can, for example, be a local area network or a wide area network and can, for example, incorporate wire, coaxial cable, fiber optic, or wireless hardware links. The communications network can provide for data encryption in some embodiments.

Server 10 can be any process that controls a resource shared by multiple processes. Without limitation server 10 can be, for example, a storage server, such as a file server or database server; an input/output server, such as a print server; a compute server or remote execution server, such as a supercomputer or mainframe that is shared by multiple users; or a process control server, such as a robot or computer-controlled manufacturing machine. Where the server is a storage server, the server can provide for data encryption in some embodiments.

Resource 11 can be anything upon which server 10 can perform operations on behalf of other processes. Thus, for example, if server 10 is a file server, resource 11 can be a file or a set or system of files; if server 10 is a print server, resource 11 can be a printer or a print queue; if server 10 is a compute server, resource 11 can be a compiler, graphics package, numerical simulation code, or any other program that the server can execute; if server 10 is a remote on-line banking computer, resource 11 can be a bank account; and so forth without limitation. Put another way, resource 11 is something that a client or intermediary can use, access, or do through server 10. It will further be appreciated that resource 11 can be one of a phirality of resources (not shown) controlled by server 10.

The bulk of the description given herein speaks as though method of the present invention. System 1 comprises client 55 system 1 comprises a single client, a single server, and one or more intermediaries, as depicted in FIG. 1A. Accordingly, the term "client" is most often used herein to indicate the process that issues an initial request, "server" to indicate the ultimate process that responds to the request, and "intermediary" to indicate a process situated between the client and the server. However, as noted above, system 1 can in some embodiments comprise multiple servers and clients, each communicating with the remainder of system 1 through communications network 20. Moreover, it will be appreciated that the terms "server," "client," and "intermediary" are in some sense relative terms, in that the same process can be viewed as a client with respect to one process, a server with respect to another, and an intermediary with respect to a third. For example, in a case in which a client issues a request to a server through a chain of several intermediaries. each intermediary other than the last one in the chain effectively stands in a client relationship to the server, and each intermediary other than the first one in the chain effectively stands in a server relationship to the client. As another example, consider once again the case in which a client, for instance a workstation, uses a print server to print a file that resides on a file server. In this example, the print 10 server is an intermediary with respect to the client workstation, and the file server is the server. Now suppose that a later time the same client workstation wants to use the print server to print a different file that is stored locally by the print server, using certain header information to be 15 supplied by a second workstation. Now the second workstation becomes the intermediary, and the print server is the ultimate server with respect to the client workstation.

In some embodiments, as is described more fully below, the invention contemplates the authentication of ACPs by means of digital signatures that can be verified by server 10. In other embodiments, to support ACP authentication system 1 further comprises an authentication server 30 that communicates with the rest of system 1 through communications network 20 via one or more secure channels 35. This is shown in FIG. 1A. Still other kinds of ACP authentication can be used within the scope of the invention.

In some embodiments, as is described more fully below, the invention contemplates two-party authentication, such that server 10 can identify or authenticate the client 5 and each intermediary 15. Authentication server 30 and its associated secure channels 35 can be included in system 1 for the purpose of two-party authentication, whether or not authentication server 30 is also used to support ACP authentication. Various other kinds of two-party authentication can also be used within the scope of the invention.

In some embodiments, as is described more fully below, the invention contemplates revocation objects associated with ACPs. In such embodiments, to support revocation objects server 10 has associated with it stable storage 12 that is accessible to server 10. "Stable" data storage is data storage wherein the stored data will survive a server failure, for example an unexpected loss of power to the server. Such storage is typically in the form of a disk file or nonvolatile medium. In embodiments where server 10 is a file server and the server's associated resource 11 is file storage, stable storage 12 can be part of, or the same as, resource 11. This is the situation shown in FIG. 1A. In other embodiments stable storage 12 can be separate and distinct from resource

#### 2.2 Requests

The present invention contemplates the use of ACPs in requests that involve a client, a server, and one or more 55 intermediaries interposed between the client and server. It will be observed that the term "request" can be used in different ways. For example, in the case of a client that makes a request to a server via a single intermediary (or, put differently, the case of a single intermediary that makes a 60 request on behalf of a client), the term "request" is used variously to refer to the client's request to the intermediary, the intermediary's request to the server, and to the overall request from the client to the server. In the case of a client that makes a request to a server via a chain of intermediaries 65 (or, put differently, the case of a last-in-the-chain intermediary that makes an ultimate request of a server on behalf of

the client and all preceding intermediaries in the chain), the term "request" is used to refer variously to the client's request to the first intermediary in the chain, the first intermediary's request to the second intermediary in the chain, and so on down the chain through the last intermediary's request to the server, as well as to the overall request from the client to the server.

In the case of the single intermediary, the term "client request" can be used more precisely to indicate the client's request to the intermediary, the term "intermediary request" to indicate the intermediary's request to the server, and the term "service request" to indicate the request received by the server, which in this case is the same as the intermediary request. This is illustrated in FIG. 1B. Client 50 issues request 51, which is a client request, to intermediary 55. In complying with client request 51, intermediary 55 issues its own request 56 to server 60. Request 56 is an intermediary request because it is issued by an intermediary. It is also a service request because it is received by a server.

In the case of multiple intermediaries, the term "client request" can be used more precisely to indicate the client's request to the first intermediary, the term "intermediary request" to indicate any intermediary's request to another intermediary or to the server, the term "first intermediary request" to indicate the first intermediary's request to the second intermediary, the term "additional intermediary request" to indicate the second or a higher-numbered intermediary's request to the next intermediary in the chain, and the term "service request" to indicate the last intermediary's request to the server. This is illustrated in FIG. 1C for the case of three intermediaries. Client 70 issues request 71, which is a client request, to first intermediary 75. In complying with client request 71, first intermediary 55 issues its own request 76 to second intermediary 80. Request 76 is an intermediary request because it is issued by an intermediary. and more particularly is a first intermediary request because it is issued by the first intermediary. Second intermediary 80, in turn, issues its own request 81 to third intermediary 85. Request 81 is an intermediary request, and more particularly is an additional intermediary request because it is issued by an intermediary other than the first. Finally, the third and last intermediary 85 issues request 86 to server 90. Request 86 is the ultimate and final request in the chain. It is an intermediary request, and more particularly is an additional intermediary request. It is also a service request because it is received by the server.

It should be noted that at any time there can be an arbitrary number of requests issued by the client or by any intermediary. An example is shown in FIG. 1D. Client 100, a user workstation, issues three client requests 101, 102, and 103. Request 101 is a request to intermediary 105, a print server, to print a file 111 that is stored on file server 110. In response to request 101 intermediary 105 issues service request 106 to server 110. Request 102 is a request to intermediary 115, a compute server, to compile a source code file 112a on file server 110 with the resulting object code to be saved in file 112b on file server 110. In response to request 102 intermediary 115 issues a service request 1160 to server 110, asking to read the source code file 112a. Later, after compilation is complete, intermediary 115 issues another service request 116b to server 110, asking to write the object code file 112b. Request 103 is a request to intermediary 120, a network server, to access a database 131 located on server 130, a database server at a location remote from client 100. In response to request 103, intermediary 120 issues a first intermediary request 121 to intermediary 125, a network server at the remote location. Intermediary 125 then issues

12

an additional intermediary request 126, which is the service request, to server 130 on behalf of client 100 and intermediary 125.

In what follows, requests are sometimes referred to as remote procedure calls (RPCs). This is because requests take the form of RPCs in certain embodiments of the invention, notably in the embodiment of the invention in a modified version of the Andrew File System as described below. It is to be understood that referring to requests as remote procedure calls or RPCs is merely a convenient shorthand that implies no loss of generality. Other kinds of request besides RPCs are very much within the scope of the invention.

#### 2.3 Delegation via ACPs

The present invention contemplates the use of ACPs in requests such as remote procedure calls that involve a client, a server, and one or more intermediaries interposed between the client and server. For ease of exposition, however, the case of an authenticated remote procedure call that does not involve any intermediaries or delegation of rights will briefly be considered first. This case, which is known in the prior art, is shown in FIG. 2A. Client Io makes an RPC to a server S, the call being of the form "op (object...)". When the server S receives the call, it must determine whether or not to execute the request. The server S proceeds in two steps:

- First, S verifies the identity of the caller I<sub>O</sub>. This it does using authentication protocols such as, for example, those described by R. M. Needham and M. D. Schroeder in their article "Using Encryption for Authentication in Large Networks of Computers," 30 Communications of the ACM 21(12) 993-999, Dec. 1978.
- Second, S verifies that I<sub>0</sub> has permission to perform this request. For example, S calls a function Check (I<sub>0</sub>, op, object, ...) which returns true or false. This check may 35 look up I<sub>0</sub> on access lists, check permission bits, or use some other means of checking authorizations.

With the case of the simple RPC thus understood, the more complex operations that are the subject of the present invention will now be considered. These operations involve 40 delegation of access rights to intermediarles. With reference to FIG. 2B, suppose I<sub>0</sub> requests some other process I<sub>1</sub> to perform an operation on its behalf. In this case, Io is termed an initiator and I1 an intermediary. According to the method of the present invention, Io creates an access control program 45 P that specifies the set of rights that it wishes to delegate to I<sub>1</sub>. The access control program is a procedure that takes as parameters a caller, the operation being performed, and the arguments to this operation, and that returns an indication (e.g., true or false) of whether or not the operation is 50 allowed. After creating P, in a preferred embodiment Io digitally signs it with its digital signature, thereby producing P>10, which  $I_0$  includes in its request to  $I_1$ . (Digital signatures are known in the art and are described, for example, in D. E. Denning, Cryptography and Data 55 Security, Reading, Mass.: Addison-Wesley Publishing Co., 1982.) The meaning of <P>0 is "Io authorizes anyone to make requests on its behalf, as long as P approves each such request."  $I_1$  then makes a request to S and transmits P>0along with the request.

The flowchart of FIG. 2C further illustrates the preceding steps, as well as the steps executed thereafter by the server S. In step 250  $I_0$  creates the ACP. In step 252  $I_0$  digitally signs the ACP to produce P>0. In step 254  $I_0$  issues its request to  $I_1$ ; P>0 is included in this request. In step 256 65  $I_1$  makes a request to S and transmits P>0 along with the request.

Before honoring the request, server S proceeds as follows:

1. In step 260 S verifies the identity of I<sub>1</sub>, the principal making the request.

- In step 262 S determines whether I<sub>0</sub> has permission to make this request. I<sub>0</sub> cannot delegate rights it does not have. This step can be carried out, for example, as a call to Check (I<sub>0</sub>, op, object, . . . ).
- In step 264 S verifies that P is correctly digitally signed by L.
- 4. In step 266 S determines whether  $I_0$  approves of the request by executing  $P(I_1 \text{ op, object, } \dots)$ .

Thereafter, in step 268 S determines whether steps 260, 262, 264, and 266 all have succeeded. If so, then execution proceeds at step 270, in which S executes the request. Otherwise execution proceeds at step 275, in which access is denied and S refuses to execute the request.

Steps 260 and 262 are the same as in the case of a no-intermediary RPC such as that depicted in FIG. 1A with one important difference: The access check in step 262 is 20 based on the initiator I<sub>0</sub> rather than the caller I<sub>1</sub>. Step 264 ensures that the intermediary has not tried to acquire additional rights by manufacturing a forged ACP or by tampering with an existing ACP. Step 266 checks that this particular request meets the restrictions imposed by I<sub>0</sub> on I<sub>1</sub> as speci25 fied in P.

The four checks of steps 260, 262, 264, and 266 are the only ones required of the server. In particular, the server does not enforce bounds on an ACP's lifetime or on who is permitted to use the ACP. Because ACPs are programs, they can check these things themselves. If an ACP is to remain valid for only a limited period of time, then it can always return "access denied" if the current time is greater than some built-in expiration date. If an initiator wishes to delegate only to certain intermediaries, then its ACP can check that the caller is one of the valid delegatees. By putting power into the ACP, the delegation mechanisms are simplified.

The purpose of having Io digitally sign the ACP is twofold. First, the digital signature proves to the server that the ACP being given to the server is the same ACP that was created and sent by the initiator Io. Second, the digital signature proves that the initiator I<sub>0</sub>, and not an impostor, actually created the ACP. This twofold purpose can also be accomplished in ways other than through the use of digital signatures. For example, in an embodiment to be described below in which the invention is incorporated in a modified version of the Andrew File System, an authentication server is used. As another example, in some systems in which a client, an intermediary, and a server execute on a single node, the node's operating system provides adequate security to ensure that the client's ACP is genuine, and no authentication as such is required. These two examples by no means exhaust the possibilities.

Deciding what operations to allow and disallow in an ACP requires that the initiator know enough about the implementation of the intermediary to give out the appropriate access rights. Note that this is inherent to the tasks of restricted access delegation and is not specific to ACPs. As an example, consider once again the delegation chain illustrated in FIG. 2C. The operation that I<sub>0</sub> requests of I<sub>1</sub> need not be the same operation that I<sub>1</sub> requests of S. Often, I<sub>1</sub> will need to perform a series of lower-level operations on S in the process of servicing I<sub>0</sub>'s request. For example, I<sub>0</sub> may instruct I<sub>1</sub> to print a file which resides on S. I<sub>1</sub> may have to make a series of calls to S in order to resolve the file's pathname, open the file, read it, and finally close it. When I<sub>0</sub> writes program P, it must be aware of the operations that I<sub>1</sub>

13

will need to make to S since P must be certain to grant these, and ideally only these operations. A later section of this description tells how  $I_0$  can avoid having to know  $I_1$ 's implementation in some embodiments of the present invention.

#### 2.4 Programming language considerations

Many languages can be used for writing ACPs, ranging from full-featured programming languages to non-procedural languages that simply list patterns to match against the arguments. Any Turing-complete language will suffice, and other languages can also be suitable. Languages containing control constructs (e.g., conditional branching) are an interesting class because they enable multiple possible paths of execution through an ACP.

Of the Turing-complete languages, simple interpreted languages such as Lisp or Tcl (tool command language) are good candidates for writing ACPs. These languages have successfully been used as extension languages to augment the base functionality of various programs. Because ACPs serve a similar purpose, namely, that of extending the core set of access control concepts that a server provides, they are likely to benefit from the same advantages these languages have exhibited in other settings. An example of using Lisp as an extension language is described in R. Stallman, GNU Emacs Manual (Pree Software Poundation, Oct. 1986); an example of using Tcl as an extension language is described in J. K. Ousterhout, "Tcl: An embeddable command language," Proceedings of the USENIX Association 1990 Winter Conference (1990).

Where a general-purpose language is used for ACPs, a concern arises with the safety of the server. A malicious or faulty ACP should be prevented from corrupting the server or using excessive resources. The server needs to be protected from program faults, such as NIL pointer differences and division by zero, and needs to enforce limits on CPU and storage use. One way to achieve this is to use a carefully coded interpreter for the ACP language. When the interpreter detects one of these conditions it aborts execution of the ACP. If an ACP is aborted, then it is assumed to have returned false, and the request is denied. Because ACPs are expected to be relatively short, strict limits can be placed on them without severely limiting their usefulness.

ACPs depend on access to request parameters and server state information to implement their checks. A core set of information that servers should provide in any ACP implementation includes the requested action and its arguments, the authenticated identity of the requestor, and the current time. Access to the requestor's identity allows checks that restrict who may make the request, and knowing the current time enables expiration checks. Provision of the current time implies the need for globally synchronized clocks. In practice this is not a problem since most modern distributed systems already employ some form of clock synchronization.

Some clients may wish to specify access controls based on additional information, such as the existence of files. Letting ACPs examine any state in the server could violate security because an ACP can transmit information to an 60 intermediary via the success or failure of the intermediary's server requests. A better model is to allow an ACP to examine any state which the ACP author could have examined through the normal server interface.

In some embodiments of the invention ACPs can also be 65 allowed to make changes to server state. Such a feature introduces a variety of problems concerning resource man-

14

agement. For the most part, a change to server state is preferably made by explicit request rather than as a side effect of an access check. One notable exception to this rule is described in the next section.

#### 2.5 Revocation

The creator of a ACP may want the ability to unilaterally and instantly revoke the rights granted by an ACP. The following is one technique for ACP revocation. The initiator writes the ACP so that the ACP checks for the existence of a revocation object, such as a file, each time it is run. If the revocation object does not exist, the request is denied. At the time the initiator creates the ACP, it also designates the revocation object. When the initiator wishes to revoke the delegated rights it simply eliminates the revocation object. All subsequent requests from the intermediary will be denied by the ACP. This revocation scheme requires no cooperation from the intermediary. It also requires no involvement by the server, except that the server must allow an ACP to inquire about its associated revocation object, and in some embodiments, that the server must create and destroy revocation objects at the initiator's request.

An important question is where to store revocation objects. If an ACP restricts access to one or more objects that reside on a single server, then that server-or, more precisely, its associated stable storage-is the ideal storage site for the ACP's revocation object. The check for the existence of the revocation object can be done locally by the server in this case. Things become more complex if an ACP can be used on multiple servers. In this case, either the ACP must be able to read a revocation object stored on a remote server, or else the revocation object must be replicated across the servers.

Another important question concerns how revocation 35 objects are designated. A straightforward scheme is to have the server create revocation objects for each client upon the client's request. The client can request creation of revocation objects at any time; however, to avoid the overhead of repeatedly having to request revocation objects, the client typically requests that a plurality of revocation objects be created ahead of time, for example when the client is initialized or reinitialized. These objects are then available to the client for later use. The client maintains a record of its revocation objects: which ones are in use in connection with particular ACPs, which ones are free and available for use with other ACPs, and which ones have been destroyed and are thus no longer available. When the client wants to create a revocable ACP, the client designates one or more of its free revocation objects by associating these objects with particular rights that are to be delegated in the ACP. The client includes code in the ACP such that the delegation of a particular right or rights in the ACP depends on the continued existence of the designated revocation object or objects. The client also updates its record of revocation objects to reflect the designation.

Still another important question concerns how revocation objects are eliminated. Typically, the client eliminates a revocation object in order to revoke the access right that is associated with the revocation object and has previously been delegated in an ACP. The client can also eliminate a revocation object for other reasons, for example, if the client decides that the ACP is no longer needed. Continuing the preceding straightforward scheme, to eliminate a revocation object the client simply requests the server to destroy the object. In response, the server authenticates the client's identity to ensure that the client is not trying to destroy someone else's revocation objects. Upon successful authen-

tication the server complies with the client's request and destroys the revocation object. The client deletes the eliminated object from its record of revocation objects.

By allowing clients to create revocation objects ahead of time, the foregoing scheme for delegating and eliminating 5 revocation objects minimizes communication between client and server. It also preserves an important advantage of the present invention, namely that the client can specify at run time which intermediaries are to have which access privileges and can do so without the need for further interaction 10 with the server. The exact implementation of the scheme depends on the particular system in which the method of the invention is embodied.

Other schemes to create, designate, eliminate, and destroy revocation objects are within the scope of the invention. For 15 example, if a server can support only a limited number of revocation objects, an allocation policy can be established to determine how many revocation objects any given client can have with respect to the server. Any of a number of such allocation policies, including preallocation of all revocation objects among clients, is possible. No matter what scheme is used, it is important that revocation objects be stored in stable storage, so that they are not inadvertently destroyed, for example through server failures.

There can be as many revocation objects per ACP as is necessary to carry out the purposes of the ACP. In particular, a single ACP can be associated with multiple revocation objects in order to provide for independent revocation of individual access rights with respect to the same request. For example, suppose client C creates an ACP that gives temporary read access to a first file called FILEONE and temporary write access to a second file called FILETWO, both of which are stored on file server S. If there are two revocation objects, one for the access rights to FILEONE and one for the access rights to FILETWO, the client can then, for example, revoke an intermediary's permission to access FILBTWO while still permitting access to FILEONE. It does so by climinating FILETWO's associated revocation object.

FIG. 2D shows the steps involved in revoking access privileges using revocation objects for another example involving FILEONE and FILETWO in one embodiment of the invention. In this example the client makes two requests FILEONE and is to be executed immediately while the second request concerns FILETWO and is to be delayed by the intermediary. The client revokes the access privileges to FILETWO before the end of the delay period, so that the second request is partly denied by the server. It is to be 50 understood that this illustrative example concerns only one of the many situations in which revocation objects can be

The steps of this example proceed as follows: It is and R2, have been created ahead of time by client C at server S. In step 280 client C designates revocation objects R1 and R2 that will correspond respectively, to delegated access rights for FILEONE and FILETWO. In step 282, the client creates the ACP. The ACP grants access rights to FILEONE 60 only for so long as the revocation object R1 continues to exist and to FILETWO only for so long as the revocation object R2 continues to exist. In step 284 the client issues a first request to the server via an intermediary L This request involves access to FILEONE. It is accompanied by the ACP. 65 In step 285 the client issues a second request that involves access to FILETWO to the server via the same intermediary

L This request is also accompanied by the ACP. The second request is a delayed request, which the intermediary will not process until a specified delay time elapses.

Next in step 286 the intermediary I, acting on the client's first request, makes a request to the server S for access to FILEONE. This service request is accompanied by the ACP. In step 288 server S executes the ACP. In so doing, the server tests for the existence of the revocation object R1 before granting access to FILEONE. If the revocation object exists, the request is performed; otherwise it is denied. At this stage of the example, both revocation objects R1 and R2 exist, so the request for access to FILEONE is performed in step 290.

Thereafter in step 291 the client eliminates the revocation object R2, which corresponds to the access rights for FILETWO. In step 292 the delay time for the client's second request elapses. In step 293, the intermediary I, acting on the client's second request after the elapse of the delay, makes a request to the server S for access to FILETWO. This service request is accompanied by the ACP. In step 294 server S executes the ACP. In so doing, the server tests for the existence of the revocation object R2 before granting access to FILETWO. At this stage of the example, R2 no longer exists, so that the ACP returns false and the request for access to FILETWO is denied in step 296.

An additional example, shown in FIGS. 2E and 2F, illustrates the use of revocation objects in a situation in which a single client request generates two intermediary requests. In FIG. 2E, client 300 is a user command process executing, for example, on a workstation. Client 300 sends request 303 to compute server 305, asking compute server 305 to compile source code file 311 which is stored on file server 310. Request 303 has an access control program that delegates to compute server 305 the rights to read source code file 311 and to write object code file 312, a version of which is already stored on file server 312 at the beginning of this example. These access rights depend respectively on the continued existence of designated revocation objects 301 and 302. In response to client request 303, compute server 305 will make two service requests to file server 310. The first will be a request 306 to read source code file 311 prior to compilation. The second will be a request 307 to write object code file 312 after compilation is completed.

Next consider what happens if during compilation the of the server via the intermediary. The first request concerns 45 user decides that the compilation request was a mistake, and that compute server 305 must not be allowed to replace the existing version of the object code file 312 with a newly compiled version. The user's command process 300 eliminates the revocation object 307 by requesting in request 315 that file server 310 destroy revocation object 302. Thereafter, when compute server requests in request 307 to write object code file 312, the ACP returns false and request 307 is denied.

FIG. 2F flowcharts these steps in greater detail. In step assumed that a number of revocation objects, including R1 55 318 the client designates revocation object R1, which corresponds to revocation object 301 in FIG. 2E, as being associated with the right to read source code file MYPROGRAM.C, which corresponds to source code file 311 in FIG. 2E. Similarly, in step 320 the client designates revocation object R2, which corresponds to revocation object 302 in FIG. 2E, as being associated with the right to write object code file MYPROGRAM.O, which corresponds to object code file 312 in FIG. 2B. In step 321 the client creates the ACP to delegate rights to read MYPROGRAM.C and to write MYPROGRAM.O contingent upon the continued existence of R1 and R2 respectively. In step 322 the client issues its compilation request to the compute server,

accompanied by the ACP. In step 324 the compute server issues to the file server a request, accompanied by the ACP, to read MYPROGRAM.C. This request corresponds to request 306 in FIG. 2E. In step 326, the file server executes the ACP. R1 exists and the ACP returns true. In step 328, the file server grants read access to MYPROGRAM.C to the compute server. In step 329 the compute server proceeds to compile MYPROGRAM.C. In step 330, the user decides not to compile after all, and accordingly the client eliminates revocation object R2 in step 332 by requesting the file server 10 to destroy R2. In step 334 the file server complies with this request and destroys R2. In step 336 the compute server finishes compiling MYPROGRAM.C. In step 338 the compute server issues to the file server a request, accompanied by the ACP, to write MYPROGRAM.O. This request cor- 15 responds to request 307 in FIG. 2E. In step 340, the file server executes the ACP. R2 no longer exists and the ACP returns false. In step 342, the file server denies write access to MYPROGRAM.O to the compute server. This completes the example.

If revocation is used often, special support for revocation objects can be provided by servers. The revocation objects are cached in memory to allow quick checks by the ACPs. The only operations on the cached revocation objects are Create, TestForExistence, and Destroy, which, respectively, 25 create, test for the existence of, and destroy the revocation objects. Only the TestForExistence operation is available to the ACP itself.

Another form of revocation is "use-once" semantics in which an ACP is automatically revoked after it is used, rather than being available for reuse by the initiator. In the printer example given earlier, suppose the initiator wishes that the printer only be allowed to read the file to be printed once. This can be supported by allowing ACPs to invoke the Destroy operation on its associated revocation objects. While this violates the principle that ACPs should be free of side-effects, permitting an ACP to destroy its revocation objects can be considered safe enough and useful enough to warrant an exception to the rule.

Still other revocation schemes are within the scope of the present invention. For example, a revocation object could be a file that is associated with multiple ACPs and that contains a list of tuples (ACP, revocation status). To check for revocation in this scheme, the ACP accesses its associated revocation object and determines whether its status is revoked. In comparison with the one-ACP-per-revocationobject scheme described above, this scheme does not take full advantage of the fact that file servers typically are optimized to check very rapidly for the existence of files and are not optimized to search through data within files.

## 2.6 Chained delegation

Now that the use of ACPs to delegate access rights to a single intermediary has been described, next is considered 55 the case of cascaded delegation among multiple intermediaries. With reference to FIG. 3A, after Io has delegated rights \_\_to\_I,\_I,\_may\_wish\_to\_delegate\_some of these\_rights\_to a second intermediary, I2. To do so, I, writes a second ACPP, ACPs to the server when it makes its request.

The flowchart of FIG. 3B further illustrates the preceding steps, as well as the steps executed thereafter by the server S, for the general case of n intermediaries. In step 350 Io creates a first ACP. In step 352 Io digitally signs this ACP to 65 produce P>0. In step 354  $I_0$  issues its request to  $I_1$ ; P>0is included in this request. Steps 355, 356, and 357 are then

executed repeatedly for values of i from 1 to n-1 inclusive. In step 355 I, creates an ith ACP. In step 356 I, digitally signs this ACP to produce <P>'i. In step 357 L issues its request to  $I_{k+1}$ ; all the ACPs created so far, that is,  $\langle P \rangle' 0 \dots \langle P \rangle' i$ , are included in this request. In step 358, I, issues a request to S and transmits all the ACPs, that is,  $\langle P \rangle^{1}$ 0... $\langle P \rangle^{1}$ i, along with the request.

When the server S receives this request, it must check, by calling Po, that Io has delegated rights to I1. Similarly, by calling P<sub>1</sub>, S checks that I<sub>1</sub> has delegated rights to I<sub>2</sub>. In the general case. S has received a call from I, along with n ACPs,  $\langle P_0 \rangle^{l}0$ ,  $\langle P_1 \rangle^{l}1$ , ...  $\langle P_{n-1} \rangle^{l}_{n-1}$ . It proceeds as

- 1. In step 360 S verifies the identity of In, the principal making the request.
- 2. In step 362 S determines whether Io has permission to make this request. This is carried out, for example, as a call to Check ( $I_0$ , op, object, . . . ).
- 3. In step 364 S verifies that each P, is correctly signed by Į,
- 4. In step 366, which is executed repeatedly n times, S verifies that I, has delegated the required rights to I,, for values of i ranging from 0 to n-1, viz:
  - S determines whether Io approves of the request by calling  $P_0$  ( $I_1$ , op, object, . . . ).
  - S determines whether  $I_1$  approves of the request by calling  $P_1$  ( $I_2$ , op, object, . . . ).
  - S determines whether L approves of the request by calling Pi  $(I_{i+1}, op, object, ...)$
  - S determines whether  $I_{n-1}$  approves of the request by calling  $P_{n-1}$  ( $I_n$ , op, object, . . . ).
- 35 Thereafter, in step 368 S determines whether steps 360, 362, 364, and all executions of step 366 have succeeded. If so, S executes the request in step 370. Otherwise execution proceeds at step 375, in which access is denied and S refuses to execute the request.

The delegation of rights along the chain from intermediary to intermediary can be represented by way of a formula. Let the initiator's set of rights be denoted as R. The set R specifies what operations the initiator is allowed to perform on which objects. Let D(I<sub>i</sub>) denote the rights delegated by intermediary I, via its ACP. D(Io) is the set of rights delegated by the initiator's ACP. Then the general formula for the set of rights obtained by the nth intermediary is

$$A(I_n)=R \cap D(I_0) \cap D(I_1) \cap \dots \cap D(I_{n-1})$$

50 This formula says that the rights of intermediary L are the rights of the initiator as restricted by all the access control programs of the intermediaries along the way (including the

Cascades of intermediaries can be more complex than the simple chain described here. For example, the present invention contemplates situations in which multiple intermediaries execute in parallel as well as or in addition to a serial chain.-Moreover, in any situation involving a cascade of intermediaries, some or all of the intermediaries can have digitally signs it and sends both ACPs to L2. L2 presents both 60 revocation objects associated with their ACPs. It will be appreciated that these example situations are illustrative and are not intended to limit the invention's scope.

### 3. A Specific Embodiment Using a Modified Version of the Andrew File System

Files represent a type of object that is widely used and for which access control is crucial. An embodiment of the present invention has been built wherein ACPs have been integrated into a distributed file system called the Andrew File System (AFS). Aspects of AFS that are relevant to the present invention are described below. A more complete description of AFS may be had by reference to the article by 5 J. Howard, M. Kazar, S. Menees, D. Nichols, M. Satyanarayanan, R. Sidebotham, and M. West, "Scale and Performance in a Distributed File System," ACM Transactions on Computer Systems 6(1):51-81, Feb. 1988. It is to be understood that ACPs according to the method of the present invention find applicability with other file systems besides AFS and with other types of servers besides file servers.

The Appendix to this application shows the modifications made to Andrew File System release 3 (as it appeared on the Ang. 19, 1990 Mt. Xinu release tape of the Mach operating system) and Tool Command Language release 3.3 in order to produce the embodiment of the present invention in the Andrew File System as described below. More about the Appendix, including copyright information, is discussed in the section "Software Appendix" that appears near the beginning of this application.

#### 3.1 Integrating ACPs into AFS

AFS is a distributed file system that operates in the context of the Unix operating system. It allows for one or more file servers and one or more clients. Each client and each server executes on its own separate node. All the clients can communicate with all the servers via a communications network. Clients can act as intermediaries with respect to one another through the use of a remote invocation facility, through which clients can execute Unix commands on other clients.

AFS supports the caching of files, that is, the local copying by a client of certain heavily used files that the 35 client obtains from a server. When a client finds a current copy of a file in its cache, it is spared the overhead of having to go to the server to get the file. AFS has a cache manager that maintains cache coherence; that is, the cache manager ensures that a client gets the correct version of a file and does not mistake an obsolete cached version of the file for the current version. For example, suppose that file FILEONE is on server S1. Client C1 reads FILEONE and caches it. Thereafter client C2 writes a new version of FILEONE to server S1. The next time client C1 reads FILEONE, AFS must ensure that C1 reads the new version of FILEONE from server S1 and ignores the now-obsolete version of FILEONE stored in its cache.

In the prior art, AFS uses a combination of access control lists and protection hits to control access to files. A variant 50 of the Kerberos protocol is used to authenticate clients. (A description of the Kerberos protocol is found in J. G. Steiner, B. C. Newman, and J. I. Schiller, "Kerberos: An Authentication Service for Open Network Systems," in Proceedings of the Winter 1988 USENIX Conference (Dallas, Tex., Feb. 55 1988) at 191-201.) The protocol that AFS uses to authenticate clients in the prior art is as shown in the flowchart of FIG. 4A: In step 410 a client Io contacts an authentication server AS. In step 412 the client Io requests from the authentication server AS an authentication ticket for com- 60 municating with a file server S. This request is made on a secure channel. (Secure channels are known in the prior art and are described, for example, in Denning, Cryptography and Data Security, supra.) In step 414 the authentication server AS creates a session key K. In step 416 the authentication server AS returns the session key K, with a ticket that has been encrypted with the file server's key K. This

ticket contains the session key along with timeout information and I<sub>0</sub>'s identity.

In the specific embodiment of the present invention that will now be described, the prior art AFS authentication protocol of FIG. 4A is replaced by the protocol illustrated in the flowchart of FIG. 4B. The format of the authentication tickets used in AFS is modified so that they can optionally include an ACP. To obtain such a ticket, a client Io proceeds as follows: In step 460, the client In contacts an authentication server AS. In step 462 the client Io requests from the authentication server AS through a secure channel an authentication ticket for communicating with a file server S. Io sends an access control program P along with the request to AS. In step 464 the authentication server AS creates a session key K. In step 466 the authentication server AS returns the session key K, with a ticket that has been encrypted with the file server's key K. AS includes the access control program P in this ticket. The ticket, when transmitted to S, convinces S that AS believed Io was the author of P. This is the signature needed to convince S to use the ACP.

Tickets in AFS are at most 2000 bytes long, which leaves room for an ACP of approximately 1900 bytes in this embodiment. Variations of the embodiment can accommodate larger ACPs through more substantial changes to the AFS protocol. There is an advantage, however, to keeping ACPs short, namely, that short ACPs are less apt to contain programming bugs than are longer ACPs, and are therefore more likely to be trustworthy. With short ACPs, the initiator minimizes the possibility that it will inadvertently give away to intermediaries any access rights other than the ones it intends to give away.

Aside from generating ACPs and including them in requests to the authentication server, no changes are made to copying by a client of certain heavily used files that the client obtains from a server. When a client finds a current copy of a file in its cache, it is spared the overhead of having to go to the server to get the file. AFS has a cache manager that maintains cache coherence; that is, the cache manager that a client gets the correct version of a file and does an ACP, it interprets the ACP to determine whether or not to grant the request.

The specific embodiment permits ACPs to be used in conjunction with the existing remote invocation facility of AFS. This facility, which is a modified version of the Unix rsh (remote shell) command, executes an arbitrary Unix command on another workstation. FIG. 4C illustrates an interaction in which an AFS client, I<sub>o</sub>, uses the remote invocation facility to run some command on intermediary I<sub>1</sub>. When the intermediary contacts the server, it presents the ticket containing the ACP. The server treats this as though the initiator had made the connection, but restricts access as specified by the ACP.

The specific embodiment of the present invention in AFS affords increased security over the unmodified AFS system. AFS's version of rsh simply passes the user's authentication ticket and session key to the intermediary in the clear. The intermediary, as well as an intruder eavesdropping on the net, can use the ticket to obtain the full rights of the initiator. With ACPs, these rights can be restricted and given short expiration times.

Variations on the specific embodiment can provide still more security by incorporating means whereby the server can independently authenticate the intermediary. For example, in one such variation the server is passed two authentication tickets, one for the initiator, which contains the ACP, and one for the intermediary. This provides additional security because the ACPs can discriminate based on the identity of the intermediary.

The specific embodiment permits an intermediary to receive or modify only data for which at least one of the client's users has access. For example, if the client is a workstation, at least one of the workstation's users must have access to the data. The embodiment does not ensure 5 that data belonging to an individual user of the workstation is protected against unauthorized access by another user of the same workstation. This follows from the operation of the AFS cache manager at the workstation. Because a workstation can be used by more than one user, the AFS cache 10 manager duplicates the permission checks that the file server performs. This prevents a user from reading a file for which he has no access rights but that is in the cache due to another user on the same workstation. However, the intraworkstation security check sometimes fails to protect data 15 properly. Variations on the embodiment can incorporate changes to the cache manager to ensure data protection for each individual user.

#### 3.2 An ACP language

The specific embodiment of the invention in AFS uses Tcl as the language for writing ACPs. Tel is suitable because it has a small re-entrant interpreter. Although Tcl delivers adequate performance, the Tcl interpreter is not optimized for speed. Other languages having faster interpreters can be 25 used in variations of the specific embodiment. Lisp is one possibility. The GNU Emacs Lisp interpreter (described in R. Stallman, GNU Emacs Manual, supra) is 3-50 times faster than the Tcl interpreter, depending on the mix of integer and string operations.

ACPs written in Tcl cannot make calls on the file system or modify the state of the server in any way. Nevertheless, they do have access to some information in addition to the requested operation and its actual parameters. This information includes a classification of the request as to whether it 35 reads or writes data, the parent directory of the file being operated upon, and the current time. The classification information simplifies ACPs since they need not list all possible operations. The parent directory makes it convenient to write ACPs that restrict access to a single directory. 40

The embodiment incorporates a few changes made to the Tcl interpreter to protect the server from malicious or faulty ACPs. These changes include a bound on the number of Tcl statements that can be executed in an ACP, a bound on the amount of storage an ACP can allocate, and a bound on 45 recursion depth. If an ACP exceeds any of these bounds, then it is terminated and treated as though it returned false, i.e., access is denied.

Following is a source code listing for a sample ACP written in Tcl that a user might want to use within the specific embodiment:

/\*Exemple 1\*/ string compare Sopolass fetch"] == 0 && [string compare \$filename "/user/joe/paper.ps" = 0 "fred"] == 0

The user would intend the program to allow read access by user "fred" to a single file, "/user/joe/paper.ps". Although the program appears at first to be a reasonable ACP, it does 65 In this program, the file id for "paper.ps" is "20000012 not actually work in this embodiment. The reasons that it does not work are described in the next section.

#### 3.3 Problems encountered in practice

Three major problems prevent users from writing simple ACPs like that presented in the preceding source code listing. Although these problems are specific to AFS, they are worth discussing since they are representative of the types of difficulties that can be encountered in integrating ACPs into existing systems.

First, an intermediary does not have an identity of its own. The AFS designers decided to avoid the substantial administrative overhead that would be entailed in establishing identities for all workstations and in handling key distribution and protection. This is not unusual in existing systems. Therefore, even in a variation on the specific embodiment in which authentication tickets can be passed to the server as described above, the server cannot require an authentication ticket from the intermediary. Thus, an ACP cannot check the identity of the intermediary.

Second, file servers in systems like AFS or NFS employ 20 unique object identifiers (ids) instead of human-readable names in most of their access requests. ACPs must refer to these file identifiers. This requirement makes it substantially harder for humans to write ACPs than would be the case if the ACPs could refer to human-readable names. Moreover, this requirement prevents ACPs from checking whether a file name matches some pattern. For example, it may be desirable for an ACP to grant access to files having names of the form "\*.o," where "\*" is a wildcard, but not to files having names of the form ".c." This is not possible in the specific embodiment since ACPs are passed file ids rather than names. A variation of the specific embodiment can be constructed in which the server provides a mapping from object id to the appropriate human-readable name for an object. However, the human-readable name for an object may not be unique since AFS allows symbolic links, that is, indirect references to files by means of human-readable names that point directly or indirectly to actual file identifiers. In this case the server must take care to map the object id to the appropriate name, probably the name under which a file was opened. Depending on the server, the information needed to decide which mapping to use may or may not be available at the time an access request is made.

Third, AFS clients are responsible for resolving a file's pathname. Each of the directories from the root of the file system to the file name must be retrieved by the client workstation. Thus, an ACP that wants to delegate read access to an individual file must also give read permission to the directories in the file's pathname.

Following is a source code listing for a valid ACP that provides read access to file "/user/joe/paper.ps":

```
/*Example 2*/
                    [string compare Sopelass
55
                            "fetch"] ==0
                     && ([string compare $fid
                            "20000012 00000d24 00001a8d"] == 0
                    | [string compare $6d
                            "20000001 00000001 00000032"]
                    || [string compare $6id
"2000000a 00000a30 000024c2"] == 0
60
                            "20000012 00001b65 00000092") == 0
```

00000d24 00001a8d". The ACP also includes the file ids for the "/", "user" and "joe" directories This ACP, which has the same intended effect as the nonworking one introduced above, works properly and actually can be used within the specific embodiment of the present invention in AFS.

Users may find it cumbersome to write ACPs like that presented in the previous program (Example 2). A tool that automatically converts an ACP like the one in Example i to one like Example 2 is thus desirable. Such a tool is outlined in a later section.

#### 3.4 Performance

To determine the overhead of using ACPs in the specific embodiment, some simple experiments were performed. The first experiment consisted of measuring the elapsed time for a stat system call under various conditions, stat is a Unix command that returns information about a file, such as the file's owner, length and modification date. The stat system call causes the AFS client to make a FetchStatus RPC to the server to fetch a block of information about a file including such information as the owner, length, and modification date. FetchStatus is an AFS command that a client uses when synchronizing its cache. FetchStatus sends a message from the client to the server on which the file is stored. The server responds by providing the client with the file's owner, length, and modification date. The client uses this result to determine whether its cached version of the file is obsolete.

Three cases were tested. In the first case no ACP is supplied with the stat call. In this case, the modifications to the AFS server have a negligible performance impact. In the second case, an ACP is supplied that always returns true. In the third case, an ACP is supplied that restricts writing to a particular directory. The source code listing for the ACP 30 used in this third case is as follows:

This ACP allows changes to the directory (first clause), changes to files within the directory (second clause), and reads to any other files (third clause). In the test, the files referenced by the stat calls were not in the directory named by the ACP, so the ACP did three string comparisons.

The test configuration comprised a Sun 4/110 running the AFS file server code and a Sun SparcStation 2 running the client code. The 4/110 is roughly half the speed of the SparcStation 2. The results of running the test are shown in Table 1.

TABLE 1

Elapsed times for stat call under various conditions.				
Test Case	Execution Time			
No ACP				
ACP that returns "true"	9.5 ms			
ACP in FIG. 7	12.8 ms			

#### 4. Coping with ACP Complexity

The general nature of ACPs makes them both powerful and dangerous. A simple coding bug can grant undesired

access rights, possibly for an unbounded length of time. The complexity of creating the desired correct ACP can be addressed in at least two ways: ACP language support and the provision of ACP templates.

A first approach to controlling complexity is through the incorporation of default provisions into ACPs. A variety of default provisions are desirable in most or even all ACPs. These include expiration times, intended users, and intended access scope. Rather than forcing ACP writers to specify these checks procedurally in each ACP, the ACP language can provide a declarative section in which the relevant values can be succinctly specified, with appropriate default values being supplied in their absence.

A second approach to controlling complexity is to place the burden of generating correct ACPs on the application and server writers instead of the application user. The application writer is more likely than the user to know the implementation details of the intermediary, and hence to know which access rights need to be delegated. This second approach, then, assumes that the application and server writers can be trusted to provide safe ACPs and that the principal culprits to safeguard against are individual instances of machines or services claiming to be something that they are not.

The print program described earlier serves as an example of this second approach. The publicly available print software (stored on a safe file server) is responsible for generating an ACP that grants access only to the file to be printed. The application writer can generate ACPs by providing a template that instantiates into a specific ACP at runtime. This template consists of an ACP that is written in advance by the application author, with slots to be filled in with usage parameters. The print ACP template can itself employ lower level templates provided by the file servers it knows about. Some example templates are:

ReadOneFile(name, lifetime, user)

restrict reading to the file given by name; access may only be by user and must occur before lifetime. Both lifetime and user can have default values

ReadOnly(lifetime, user)

restrict rights to the user's read rights.

WriteInDir(directory, lifetime, user)
restrict writing to a single directory

45 When one of these is instantiated, it becomes an ACP with the arguments hard-coded in.

#### 5. Additional Aspects of the Invention

Several additional aspects of the invention will now be 50 presented. These aspects are presented by way of illustration and are in no way intended to limit the scope of the invention

ACPs according to the method of the present invention are useful for delegation in other settings beyond the context of distributed systems. One such setting is a system in which client and server are processes running on the same computer or workstation rather than on two or more physically separate computers. For example, consider a workstation user that has just received a large program from a mailing list and wants to run that program. The user is not completely sure he can trust this software; for example, the user is concerned that the software may be infected by a software virus, or that it may attempt to execute another program without the user's knowledge. However, the user knows thich files the software should write and execute if it works correctly. According to the method of the present invention, the user creates an ACP and attaches it to an invocation of

the program. In this aspect of the invention, the initiator is the user's command processor, the intermediary is the untrusted software, and the server is the local operating system. Because the ACPs can be managed securely by the operating system, digital signatures are not required.

Another example of the use of ACPs in a nondistributed system concerns setuid programs. Many operating systems have a facility that allows a user to delegate his rights to a program known as a setuid program. When the setuid program runs, it executes with the rights of the owner of the 10 program file instead of the rights of the invoker. A typical Unix system, for example, has many of these programs that run as the so-called super-user, who has unlimited access rights. Bugs in these setuid programs are a major source of such a program to cause it to read, modify, or execute some file that the program author did not intend it to reference.

An ACP attached to a setuid program can limit the damage caused by a bug therein. In principle, the attachment of the ACP provides no additional protection since the 20 system administrator is responsible for both the setuid program and the ACP. In practice, however, the ACP can be made much simpler than the setuid program. This simplicity have a damaging bug than the setuid program alone.

ACPs can be used to provide a limited form of setuid functionality in a distributed system. A game program running in a distributed system can have an embedded ACP that 30 allows any user to modify the high score file. When the game wishes to modify the score file, it uses this ACP, running as whoever invoked the game. In principle, an intruder could look in the program, decode it, and extract the ACP. But if modify the high score file for this particular game. The alternative for a network setuid facility would be to encode something equivalent to the author's full rights, such as his password, in the program. Use of the ACP limits the damage if the program is decoded. For certain applications, this can 40 provide adequate security.

#### 6. Conclusion

Access control programs (ACPs) permit controlled delegation of access rights to untrusted computer hosts. If an 45 initiator of an action does not fully trust an intermediary, then it can create an ACP and pass it to the intermediary. The ACP is executed at a server for each request made by the untrusted intermediary and decides whether or not to allow

the request. An ACP cannot grant rights that its creator did not possess; it can only restrict these rights.

ACPs can be incorporated into current systems to augment such systems' existing access control facilities. Typically this incorporation requires only digital signatures, two-party authentication, and an interpreter for the ACP language. The exact implementation in each case depends on the specifics of the underlying system. One of the strengths of the ACP design is that it places no assumption on the type of access control provided by the system or on the authentication protocol employed. ACPs can be used with both public and private-key based security mechanisms.

All decisions about the rights delegated by an ACP are incorporated into the ACP itself, including who is allowed to security weaknesses. A malicious user can exploit a bug in 15 use the ACP and for how long. Because ACPs bear the power of a full programming language, a wide range of access control policies can be implemented by clients without the involvement of servers. Because ACPs are fully contained and digitally signed, they can be freely passed around. Typically, a client will pass an ACP along with an operation request or with a connection setup request. However, ACPs can also be published, in which case, a request need only identify which ACP authorizes it.

The power of ACPs must be used carefully. In particular, bination of the setuid program and the ACP is less likely to

25 servers must protect themselves from malicious or buggy ently delegating excessive rights. Servers can protect themselves by using a carefully coded interpreter. The use of templates and ACP language defaults as described above can aid the construction of correct ACPs.

A general problem of restricted delegation is that the initiator must know enough about the implementation of the intermediary to delegate the appropriate set of access rights. Clients tend to think in terms of high-level operations, such he did, all the intruder would get would be the ability to 35 as "print this file", while the accesses made by intermediaries to servers involve lower-level operations, such as "stat this directory then retrieve the file with this unique ID." ACP translation tools-that is, software tools that translate an ACP program expressed in a form convenient for the programmer into an ACP program expressed in a form usable by the server-can be written to provide help in coping with this

> Although the above is a complete description of certain embodiments and aspects of the invention, various alternatives, modifications, and equivalents can be used. Therefore, the above description should not be taken as limiting the scope of the invention. Rather, the scope of the invention is defined by the appended claims along with the full scope of equivalents to which these claims are entitled.



XEROX D/92140

# nichols

Document name:

Diffs

Printing date/time:

June 2, 1993 11:45:57 am PDT

Host name:

osprey

This file, "Diffs," represents the differences in source code between release 3 of the Andrew File System (as it appeared on the August 19, 1990 Mt. Xinu release tape of the Mach operating system) and a modified version of release 3 of the Andrew File System, described elsewhere in this application, that embodies the method of the present invention.

The modifications made to Andrew File System release 3 (as it appeared on the August 19, 1990 Mt. Xinu release tape of the Mach operating system) in order to produce the embodiment of the present invention in a modified version of the Andrew File System as described elsewhere in this application represent original unpublished work of Xerox Corporation. This original unpublished work is Copyright \*1991 Xerox Corporation. All rights reserved. Copyright protection claimed includes all forms and matters of copyrightable material and information now allowed by statutory or judicial law or hereafter granted, including without limitation, material generated from the software programs which are displayed on the screen such as icons, screen display looks, etc.

nichols

हुम्पूर्क क्रिकेट क्रिकेट क्रिकेट

C05d: Mon May 24 12:24:03 PDT 1993

Job #588



--- 13,18 ---INSTALL-4 (SECOLED DIS/INSTAL)
COMPILE\_RT - 8 (SECOLE) DIS/OSSOIIS\_ET

dedice numrecricariem (190-1814) (180)

dedice numrecricariem (190-1814) (190

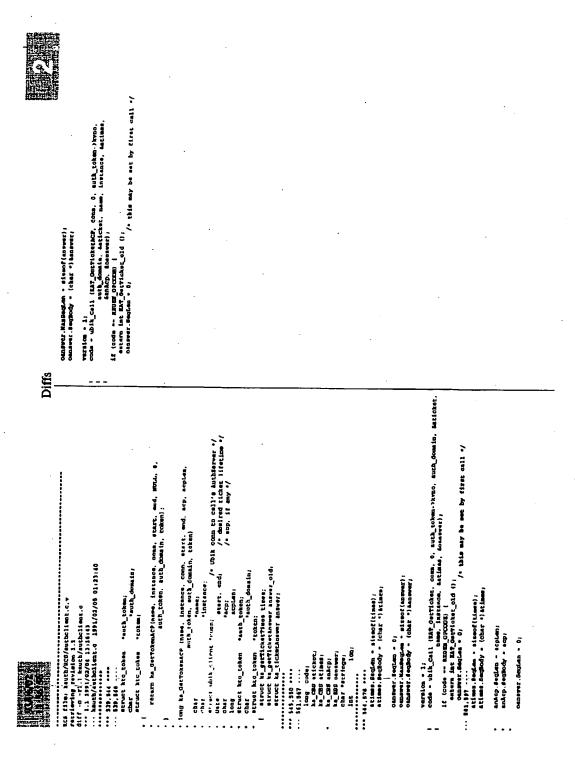
CFLAGG = -g -18 (SECOIR) include \$ (Alectra)
INCLG-\$ (SECOIR) includes (Abit. b.)
(SECOIR) includes (1 to b.)
8 (SECOIR) includes (1 to b.)
8 (SECOIR) includes (1 to b.)

#define maintentainin #define itc\_ting\_unchitaint [15-60]

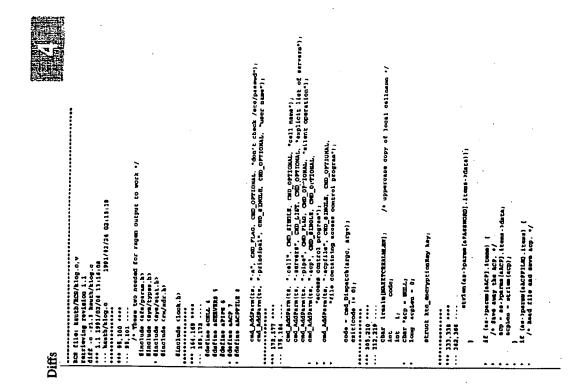
CFLAGS - . g - 16 (SRCDIR) include ((AIRCOPTE) . Butatio INCLS-6 (SRCDIR) include/ubik. h \

klog: keuth.h kantile.h libbauth.s filiss; krb\_tf.c klog.o co ·o klog klog.o kib\_ff.o libkauth.s filiss) GG .o kes kas.o admin\_tools.c libkauth.a \$ [LIBS]

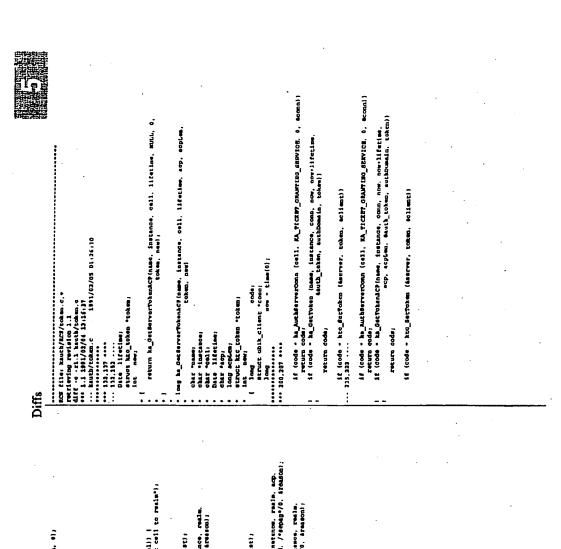
01/09/2003, EAST Version: 1.03.0002



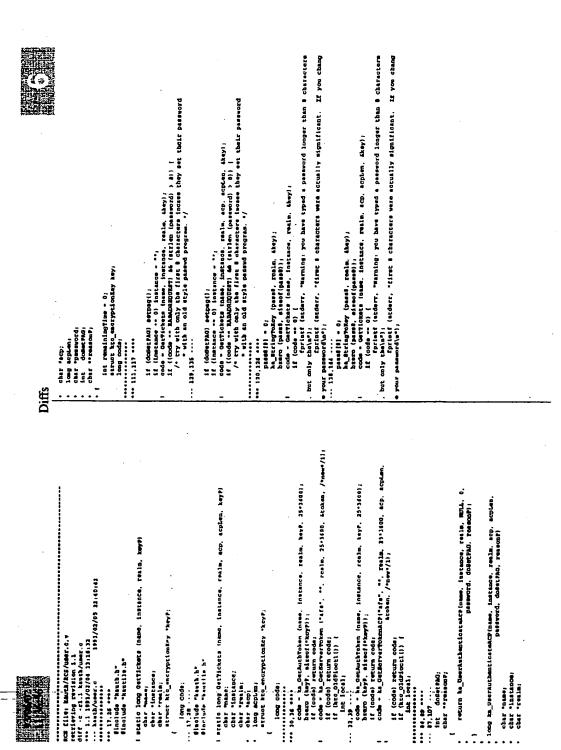
	<pre>ba_cas</pre>	) Thom far_defricter (call, kvno, authOrasin, atlonet, namno, sinstance, stimes, oanswer)  1377,1566 **etises) /* encrypted start & end time */  14.018 **enasses;  14.018 **enasses;  1	long tar detrictet (call, kvno, authDomain, aticket, Chame, ainstance, atimes, deads	Inde Ilai, settles (cell), version, schie_accounts, statics, dynamics)  **List	Na_CSS *ational seasons!  Obs. *ations: "sinctoon;  Na_CSS *ations:   "excrypted start a mid time */  Na_CSS *ations *ations */  I trium detrichet (4, call, kvno, atthomain, atichor, ansas, sincteoon,  ations, acy, cansers);	long The destruct (call, version, admin_acocunts, streics, dynamics)
Diffs	9191	1 oker	SETUCE FE_Call Feal); Jang SETUCE FE_CALL   Avao; FERUTH COde;   static long depticher (werston, cal), Avao, sutbbonsin, aticket, Gname,   dat version; elimetance, stimes, acp. ososswat)   dat version;   call;   veal;   ve	1406,1403 *mnase; dhar dhar *atiss; daring start 4 and tiss */ th. CHS *atiss; actiss; b. CHS * actiss* actis*	code . tk_Watericto! (!ickmt . d. ichntinn. ideavor.kby.  (a) idea-unorin, hamme, callet .useriD.inntence, cell.  (i) it idea-unorin. d. deselondey.  I joedo! joto abort.  switch (warston) (  1509,1331  1509,1331  thas. start-otch1 (sater-unoriD.indtance):  thas. start-otch1 (sater-unoriD.indtance):  thas. start-otch1 (sater-unoriD.indtance):  thas. start-otch1 (sater-unoridetiae)));	code - tht_materictettes. Astebettam, seerwar.key.  Calar.useribl.mas. calar.useribl.mas.call.  tlass.ast.t and, desembonky.  ra_boscottonky.  ra_boscottonky.

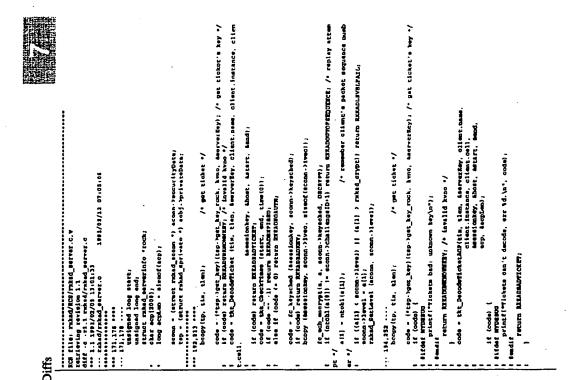


Lidel keuthy fact/hauch.rg.v
Lidel keuthy fact/hauch.rg.v
Lidel hatth/hauth.rg
Lidel hatth/ha

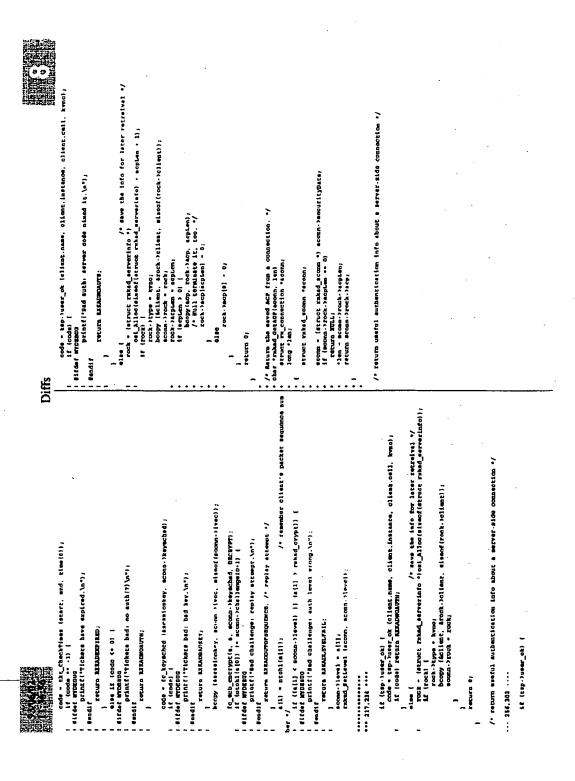


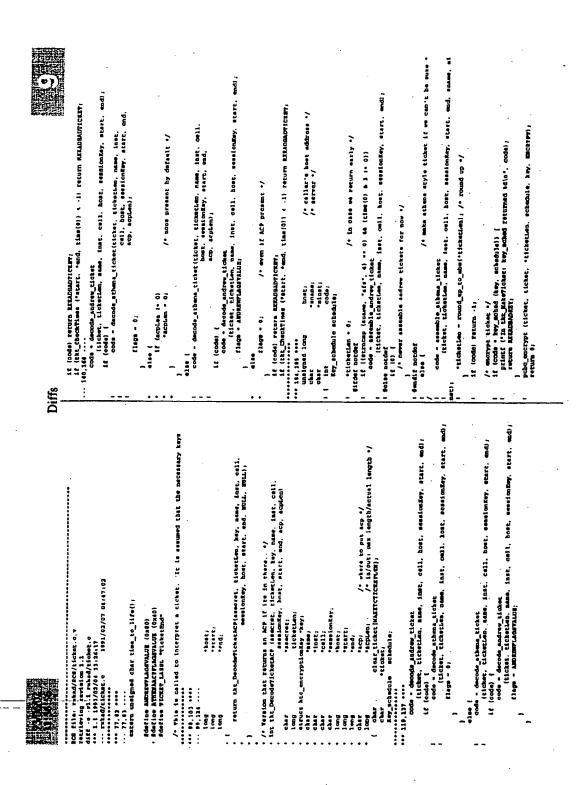
--- 370,377 ---

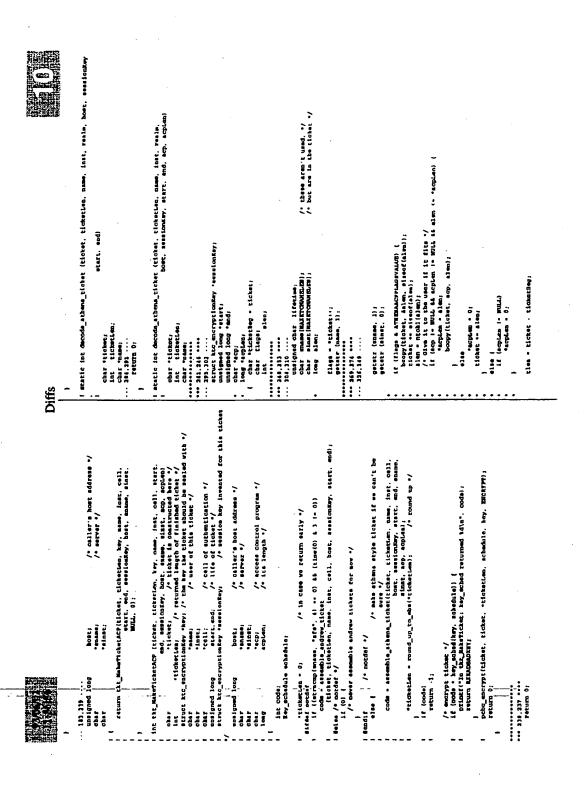


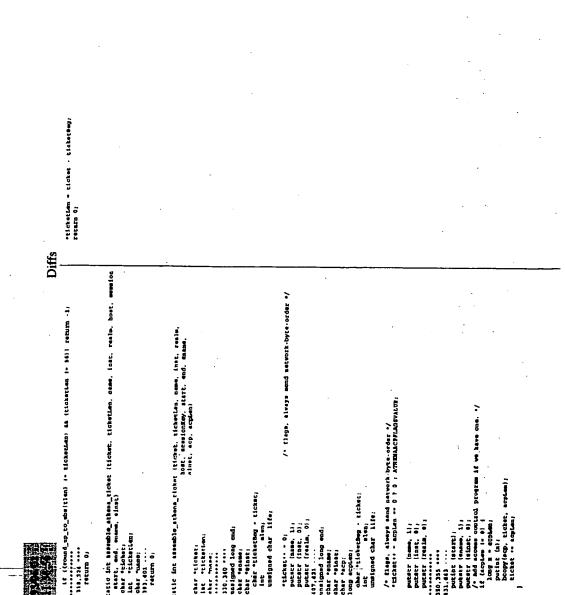


MCS Elle: rand/sci/masfile; vererie-ing revision 1.1 dies of the control of the c

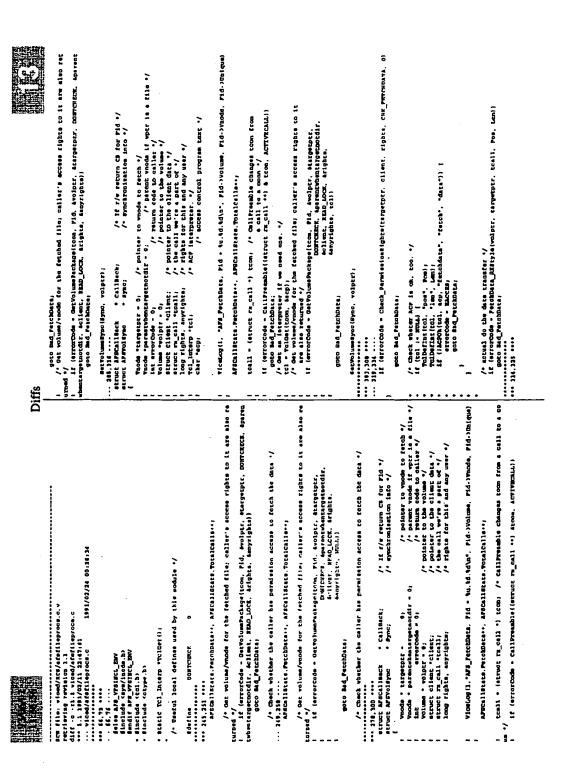




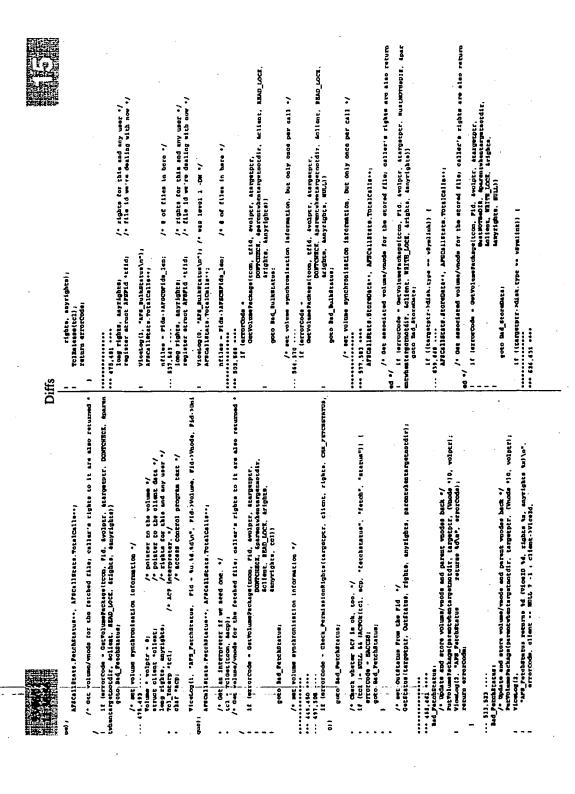








If Constitution (1992)   Activation (1992)	Comment of the control of the cont	Di	S. S.
pero Bad_parebad; bollocit; and parebad; bollocit; wolder, tollocit; wolder, and parebad; bollocit; wolder, tollocit; wolder, wolder, tollocit; wolder, wold	medical termination of the company o	<pre>1f (Volumedriltaeble (volgari)</pre>	(errorcode = OetVoluseFackage(tcon,
serolumary of pro. volper);  serolumary of pro. volper);  if introduced a lack of the control of	serolumary of pro. volper);  serolumary of pro. volper);  (**)  ******************************	9	DON'TOTAGE, sparentubentargetzoedit, dallane, Rado, Loce, arights, Americhes, east
### ##################################	### STATE OF THE PARTY OF THE P	Purvolume Packaga (parentwhentargetnotdir, targetper, (vande +)0, volper), Vicalog(2, "APE_Fetchidta returns %d\n", errorCode);	
		callPottable(tcm); return(srotOds);	SetVolumeSymo (Symo, volukk);
(* Check Water Act 16 of, too. */   (* Out the Access Life from the dirs whole */   (* Out the Access Life from the dirs whole */   (* Out the Access Life from the dirs whole */   (* Out the Access Life from the dirs whole */   (* Out the Access Life from the dirs whole */   (* Out the Access Life from the dirs whole */   (* Out the Access Life from the dirs whole */   (* Out the Access Life from the dirs whole */   (* Out the Access Life from the dirs whole */   (* Out the Access Life from the dirs whole */   (* Out the Access Life from the dirs whole */   (* Out the Act 16 out the Access Life from the dirs whole */   (* Out the Act 16 out the Access Life from the dirs whole */   (* Out the Act 16 out the dirs whole for the feathed file; oiller's access rights to be access from the dir whole */   (* Out the Act 16 out the dirs whole */   (* Out the Act 16 out the dirs whole for the feathed file; oiller's access from the dir whole */   (* Out the Act 16 out the dirs whole for the feathed file; oiller's access from the dir whole */   (* Out the Act 16 out the dirs whole for the feathed file; oiller's access from the direct Children's acc	Comment water Art 18 of, too, 'fetched', 'fetch', 'adl' )	Constitut	196,401
14 (101 - 10	intect;  (**) (**) (**) (**) (**) (**) (**) (**	ari coable (voluer)	s. Terroroms . Learn_germiseromsgure(tergerpts, citomt, fights, CHK_grcKACL, 0)) goto Bid_PetchAct,
inde *) 0, volpt?;  (*) dot the Access List from the dir's wode */  If (accode * Effects Accessisatisty part, parentylentatgemotdir, Accessisatisty part, parentylentatgemotdir, Accessisation */  ***Intercolous ***Int	processing streeting and present the form the dir's whose ',    dots and present the formes that from the dir's whose ',    formed and present the formes that from the dir's whose ',    formed and present the formes that from the dir's whose ',    formes and present the formes that the formes th	GettallBathStrutt (AddCallBath(clites: 'bbost, Pid), CallBath;	/* Check whater ACP is ob, too. */
( Out the Access List from the dire woods '/     ( Out the Access List from the dire woods '/     ( Out the Access List from the dire woods '/     ( Out the Access List from the dire woods '/	ide 1 0, volptri)  1 ( oct the Access List from the dir's woods ')  11 ( according a Rivel Accessisting to the control Accessisting to the control and a second a recently a return a second a recently a return a second a second a return a second a second a return a second a se	5	errorde - Backs;
/- Opt the Access List from the dirty woods //   Serence   If (arronded - Mr. Parcol, Accessisettarypettr. parametrsencedtr. Access   Serence   Arronded - Mr. Parcol, Accessisettarypettr.   Parcol, Accessisettarypettr.	( der the Access List from the drive woods ")	(Wnode	Internative form cook
# 1970 Bill Problems   1970 Bi		CallPostable (total)	/* Oot the Access List from the dir's wands */
mus */  Purvolums Package [parent panet registed of transport. [Vende *10, volpti]  21) Call Dogstable (*Neg_Package returns *14 (Act-1s) [ur', strontode, Accessinge.*) follows and the control of the c	Purboluma Package   Purb	return (errarthde);	goto Bul PatolMCLi
PurbolumePackage prentratestalfr.tirgstpdr.(fronde *10, volptr);   VolumePackage    Value	Purblusselding principal profession   Purblusselding presentific timpstport (foods *10, volppt)	READS_PetchAct (tcon, Pid, Accessifiet, Outstatus, Sync)	3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ima */  * Tribotable (tool)  * Tribotable (tool)  * Tribotable (tool)  ** Tribotable (to	inde */  * Tribatessifeti.  * Tribatessifeti.  * Tribatessifeti.  * Tribatessifeti.  ** Tribatessifeti.  *		
int date */ int date */  yeturn errorcode;  yours tout '-/  ne. 71d.:Nonde, Fid.:Unique)  ***********************************	int data */  int are reproduced.  ACTIVEOLL)  ACTIVEOL	volptr - 0; /-	
ACTIVECALL)  *** 71d.: Whode, Fid.: Uhique)  ***********************************	ACTIVECAL.)  ACTIVECAL	•	TCIRoless(tCl); return errorCode,
ACTIVECALA)  ACTIVECALA  ACTIVECT  ACT	ACTIVECALL)  ACTIV	/- ACP 1	
ACTIVECAL.)  ( Our volume/vonde for the factabed file; online; accepter; the factabed file; online; accepter; the factabed file; accepter; the factabed file; accepter; the factabed file; accepter; the factabed; the fac	ACTIVECALL)  ACTIV	714	. 187, 188
ACTIVECALL)  1 If (Extremed a carvolume and tile; oaller; active to the area of the collect.  4 If (Extremed a carvolume and processes, and the souther; assembler; controller;  4 If (Extremed a carvolume and a carvolume) and any sequence of the carvolume and carvolume and the carvolume ana	ACTIVECALI)  1 di (Entrocode derrollumentargetoca, 71d, % objet; stargetot; DONTORICE, whentystrocedi; dellan, mano_Loca, arighte, sanytighte)  4000 Bial_Fecchali,  410,431  410,431  410,431  420,431  430,431 .	sts.Petobact	Afforthere.vetthach Afforthere.fotalCalle
ACTIVECALL)  in 18 (errorcode - OstvolumeNachapeltom, Pid, Svolptr, stargetptr, DOSTCHECK, wherever, and the stargetptr, DOSTCHECK, and the stargetptr, DOSTCHECK, and the stargetptr, of the stargetptr of the stargetptr of the stargetptr of the stargetptr, of the stargetptr of the stargetptr, of th	in til (errorcode - GarvolumeNariage(tom, Fid. évolper, stargetper, Dorrcherer, chemical geoco Bag, Perchadi, solidem, Rabo_Goos, arighte, sanyidate);  (**Check Warther we have pormission to forch the Act. *)  (**Check Warther we have pormission to forch the Act. *)  (**Check Warther warther we have pormission to forch the Act. *)  (**Check Warther warther we have permission to forch the Act. *)  (**Check Warther warther we have permission to forch the Act. *)  (**Check Warther warther we have permission to forch the Act. *)  (**Check Warther warther we have permission to forch the Act. *)  (**Check Warther warther we have permission to forch the Act. *)  (**Check Warther warther we have permission to forch the Act. *)  (**Check Warther warther we have permission to forch the Act. *)  (**Check Warther warther we have permission to forch the Act. *)  (**Check Warther warther we have permission to forch the Act. *)  (**Check Warther warther we have permission to the Volume *)  (**Check Warther warther warther we have permission to the Volume *)  (**Check Warther w	••• 142, 342 •••	Oet volume/vmode for the fetched file; oslier's addess rights to it are
erpripts to it are also se imperptr confidence, bearen imperimentally acriverally)	ore Dad_Perchali,  (135,131  APRILIBEAT OF EAR APRILIBEATE. PERCHALLE, AFFICE  APRILIBEATE. CONTORER, Aparen  LINE () Chet volume/mode for the form  APRILIBEATE.  AP	<pre>if (errorcode - callpreamble(struct ru_call **) &amp;tcon, ActiveOALL)) goto bad_PetchActr</pre>	*terpetptr, DOSTONECK,
rangetptr. CONTCHECK, Aparen umed  Activecaci)	*** **********************************	Accessis: - AFPOSeque_3 no - 0; Accessis: - AFPOSeque_41 - malloc(AFPOPACHRAX);	goto Bad_PecchAcL; /* Chech whetbor we have porrejeston to fetch the AcL */
ACTIVICALL)	ACTIVENAL)  ACTIVENAL	/* Oet volume/woods for the fetched file; deller's access rights to it are also re	AND CALLES MANCHANGES APPROXIMATION OF COLUMN SANGERS
Perrogation (1777)	ACTIVELALA)  ACTIVELALA  ACTIVELA  AC		
### SetVolumeDyna (Sync, volptr)   1	### ServolumeSpac (Spac, volpt7);  16 (arrancode = Calibreand)s((struct rr_dall **) & becoo, ActiveCALL))  9000 Mad_FetchAct;  9000 Mad_FetchAct;  17 (arrancode = Calibreand)s((struct rr_dall **) & becoo, ActiveCALL))  18 (arrancode = Calibreand)s((struct rr_dall **) & calibreand)s((struct rr_da	when targethodir, belief, Rid Lock, brights, darrights) pere bed petchat.	(* cet volume/whose for the fetched file; caller's access rights to it are also ret med v/ 12 (errorCode · GetYolumeRackage(toda, f/d, 4volptr, stargatptr,
if (errorcode - calibremable(ferror re_call **) &tooo, ACTIVECALL)) goto Bad_PetchAct, goto Bad_PetchAct, /* Check whether we have permissio /* Check whether we require a check whether we have permissio /* Check whether we require a check whether we require a check whether whether whether we require a check whether whether we require a check whether whether we require a check whether whether whether we require a check whether whe	if (errorchde - Callpramable((struct rr_dall **) &cooc, ACTIVEDALL))  goed Bad_BeachAda,  goed Bad_BeachAda,  /* Check whether we have permi  col = Tight((coc, Acp);  Accessifet-NAFOscow_les - 0;  Accessifet-NAFOscow_les - 0;  Accessifet-NAFOscow_les - 0;  Accessifet-NAFOscow_les - 0;  Accessifet-NAFOscow_wis - anito(Argonagas);  Accessifet-NAFOscow_wis - anito(Argonagas);  /* Cost volume/whode for the facched file; caller's access fights to it are also ze  ViceLogii, *Arg_Petchetue,	SetVolumeSyac (Syac, volptr);	powickick, apermitwhenersquincies, endir, endire, endire, endire, endire,
/* Opt in interpreter (d we need one, */  tol * Tolder(teos, dacp);  Accessilat.>AF80paqua_los 0;  Accessilat.>Accessilat.oo 0;  Accessilat.oo of control of co	/* Get to isrepreted (if we need one. */  tcl - fclocitcos, decp);  Accessifist-NAFOpene les · 0;  Accessifist les · 0;  Accessifiet les ·	Dde • CallPres	
McGessilst-3A750paque les - 0; Accessilst-3A750paque les - 0; Accessilst-3A750paque wil - malloc(a760paqumual); Accessilst-3A750paque wil - malloc(a760paqumual);	Accessifet.hFforeque_ies - 0; Access	/* Oet ub interpreter (6	lether we have permission to fatch the ACL
loog rights, anyrights;	long rights to it are also re Vicologii, 'AFE_PatchBistue,		olptr = 0; bot *ollent;
	the feetched file, celler's access rights to it are also ze vicelogii, 'AFE_Petchstatus,	ACCOUNTER COATON ONLY OF MAILOC (APROPACITALE);	

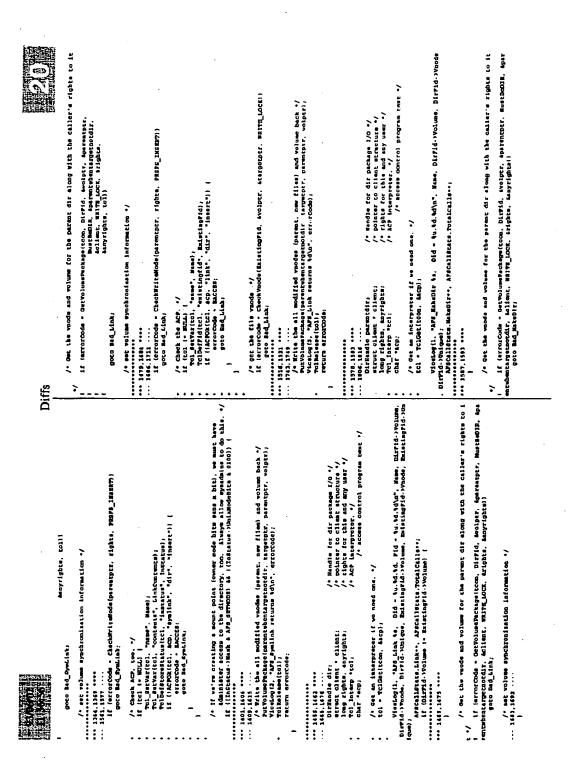


Diffs	callrostable(toom); return(errortode); } 337,743 electrostable(toom);	::	0-1	/ Chack if we have permission to chungs the dir's ACL */ 781,780 APSCALIState.Storewick.*, AFSCALIState.Totalcalls**; /* Obt associated volume/woods for the target dir; caller's rights are also returned	d -/ if (errorcode - detVolumePackage(iccn. Fid, bvolpti i hastbadis, dorrantsi elident, HilffLiock i goto Bad Storakci, sanyrighte, Will.)	/* Chock if we have permission to change the dir's ACL */ ***********************************	if (errorcode - Calibrambie(ferrorc Fr_call **) atcom, ACTIVECALL)) good Bad_stormAct;  * /* Our am interpreter if we need one. */  * tol = Toloetitoon, angp;  * ViceLogii, *ARE SCORACL, Fid * to.td.td, ACL*te\n", Fid->Volume, Fid->Vnode, Fid-> Toloetitoon, Accessisters. Accessisters. Accessisters. Accessisters.	*** 144.750 ****  *** 144.750 ****  *** Indeptus Mash * 0;
	direct coll tendii; // reasobar the call structure for ttp */ struct roor rights, amprights; // rights for this and any user */ int   newinode 0; // Arc interpress; // those of men'y strated version */ ober **sip; // Arc interpress; // ober interpress; // ober **sip; // coress ontrol program tent */	Vicatop 1. 'AFS_StoreData. Fid * Nu.td.Ad\na", Fid.>Volume. Fid.>Vnode, Fid.>Under.)  ***********************************	ned */ If legrorCode * OstunianPackage(toon, Fid, &volptr, &targetptr, AustHOTBeDIR, &parententgetpot(dir, &cilinnt, wRITE_LOCK, &righte, &sayrighte)) goop had Eternia; //*.mer volume myschromisetion infozaation */	102.715  If derivations - callpramable ((struct rg_osl) **) (troo, Activation))  grob led_alcordits;  /* One is interpreter if we need one. */  tcl * folloritons, seeps!	** One desociated volume/wadd for the sturred file; caller's rights are also returned for detailers of the sturrence of detailments of the sturrence of detailments of the sturrence of the study of the st	goto bad_btorchita.  /* set volume synchronistion information */  *** 114.744  124.744  Grecorcodo - Check_PermissionRights(targetptr, client, rights, CML_SYDROMYA, GOOD   GOOD		/· this bit means that the locks are set and protections are OK */ fra_setLoosStatus(teall, 1); ************************************

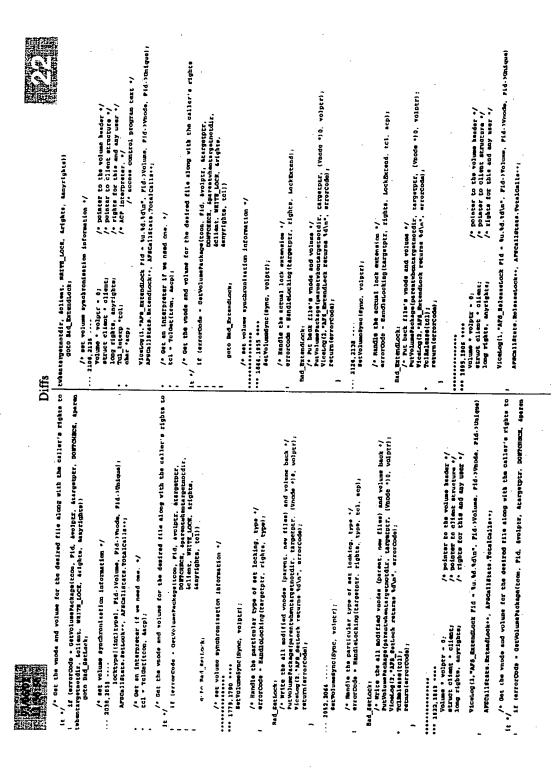
objection information */  if not no interpreted its called an attack and the following of the target of target of the target of the target of the target of the target of			
wigother are also return us); APPGAISEREA SCORRETURN. PIG APPGAISEREA SCORRETURN. APRCED RESPONSE STATEMENT OF THE EXTREMENTAL OF THE VOICES OF SETVILLES FOR EXPENSE STATEMENT OF THE VOICES OF SETVILLES STATEMENT OF THE VOICES OF SETVILLES FOR EXPENSE STATEMENT OF SETTIME STATEMENT OF SETTIMES STATEMENT OF SETTIMES STATEMENT OF SETTIMES STATEMENT OF SETTIMES SETTIMES STATEMENT OF SETTIMES STATEMENT OF SETTIMES SETTI	r volume synchronisation	` }	M peed one.
restoring.  Trestoring.  Tresto	sted volume	target dir, caller's rights are also return	ViceLog(1, "AFG_StoreStatus, Fid
Triples, CTL_STORAGE, AZ 10 1	•/ 1f.(errorCode = DetVolue	Fid. bvolptr, Stargetptr.	AFFCallState.Storestatus**,
t. rights, CRT_STORESCI, at to the state of the state	Mustbedi	R, Aparentybentargethotdir, Maitwijock, arighte,	/* Cht volume/whode for the target file; caller's righte to it are also if (errorcode - detTolumePackage(tcm, Pid. Avaluer, Asarcaener
t. rights, CH_srowshel, al and		(ts. (cl))	DOMPCHECK, Sperentebentergernoedly, schlent, warre lock, secoles.
The section of the se	luse epsebronisation		٠.
t. rights, CHL_STOREACE, 62  "eCl.)			
c. rights, CHL_sTORARCE, 62 10 10 10 10 10 10 10 10 10 10 10 10 10	850,061		/ set voices specifical setting information //
to the state of th	16 (errorcode - check_Permismichtsinghtsingtelws))	targetptr. client, rights, CRE STORBACE, &1	*** 805,014 **** *** 921,935 ****
cde *10. volper); } to a comment of the comment of			If (extercode - Check_PermissionRights(targetotr, client, rights, CHK_(satestas))
dd ")0, wolper); } to a see a	. /* 'The't whether ACP is ok, too, e/	Orderol", "ertore", "acti") (	goto had Storestatus,
dd *10, *0]ptr); } to a see a	octofode Electry		
dd *)0, vojper); } to *  **Code *)0, vojper); }  **Code *)0, vojper); }  **Code	-		•
ode *10. volptr); % to s  was header */ kructure */ dima, Fid.:Wrode, Fid.:Ont stperptr, Conventmed *  stperptr, Conventmed *  stperptr, Conventmed *  stperptr, Conventmed *  dimy user */ dimy user */	/* Build and store the new Accoss List	for the dir */	if (iACPOh(rol, scp, "storestatus", "store", errortode - BACCES;
and a '10, volptr); % to see the bander ', % to see the bander ', % % % % % % % % % % % % % % % % % %	<pre>1f ferrorCode * RISLore_Accessist(targ goto Bad_StoraACL,</pre>	etper, Accessifet))	
me beader ', tructure ', dume, Fid.: Whode, Fid.: Ont a to it are also returned ' stperptr, CoursonClick, Sparso ode, D dury user ', dury user ', dury user ',			
mes beader '/  tructure '/  d any user '/  speciptr, Dorregingl, Spares  charters '/  structure '/  structure '/  structure '/  d any user '/  d any user '/  structure '/  d any user '/  structure '/  d any user '/	672, 677		/* obeck for a symbolic limk; we can't chand these (otherwise could the
wer beader '/ d any user '/ olume, Fid. Mande, Fid. 30h1 s to it are also returned ' siperptr, CONTGREE, Spares d any user '/ d any user '/ d any user '/	PutVolumePackage(parentwhentargetnotdir Vicelog (2, "APS StorwACL returns 14\n",	erroccode);	to a at pt of vice versa) ./
were bander ' , dany user ' , b olume, Fid. Mande, Fid. Man			errorcode a student,
were beader . / d any user . / olume, Fid. Monde, Fid. Mon. stylett, Donroffect, Sparen ode, D d any user . /	return errescode;		028 033
d any user '/	-		949,935
we header ',  tructure ',  dum, Fid. Monde, Fid. Mn1  a to it are also returned '  superpir, CONTGREE, Spares  der beader ',  dimy user ',  dimy user ',	***************************************		:
dany user '/  olume, Fid. Mande, Fid. Non.  s to it are also returned  sigetptr, Donroffect, Sparen ode, D  structure '/ dany user '/ dany user '/	olptr - 0, /.	volume header	StoreStatus returns (d)n", errordos);
olume, Fid. Mande, Fid. Mail  s to it are also returned * sigetptr, Donroffect, Spares  ode, D  structure */ d any user */		gate for this and any user ./	
a to it are also returned . signingt, teares ode, D strotture ./ d any user ./	Vicelog(1, "APS_StoreStatus, Fid	1d.1d/n", Fid.)Volume, Fid.)Vnode, Fid.>Uni	
argetptr, CONTGREE, Spares ode, D structure / enterer / enterer / enterer / enterer / enterer / enterer /		s.TotalCalla;	dir. /-
stpetptr, DORNGREE, Sparen ode, D ode, E of the structure		; dalley's rights to it are also returned "	: د
/* set wolume syndhromisation information */ 897.51: ************************************	if (errorcode = detvolumePechage(toos, twhentargetootdir, dollent, White_lock, Arigh goto Mad_storastatus)	914. dvolptr, ktargetptr, DONTGREE, Sparen to. danyrightell	Vicelog(1. *AFE_Resoverile te, Dirfid-Jühique); AFECallState.Esmoverile:, AFEC
Volume • volger • D; /* pointer to the volume header •/ emittee error olient • olient; /* pointer to olient erructure •/ /* regite for this and may user •/ /* regite for this and may user •/ /* regite for this and may user */ /* regite for this and may user •/ /* regite for this may user •/ /* regite for this and may user •/ /* regite for this may u	•	<i>/•</i> 6	
fights; /' rights for this and any user "/ /" set volume synchronisation into	olptr - 0; /-	inter to the volume header ./	i (strontous = detvolumePstagmitten, diriid, švojptr, šparantptr, Mai unisbonistrotuli, scilmit, Wilfi_LOCK, hrights, kanyrights) gote Bad kasovepijs,
/* Act laterpreter. */	•	ghte for this and any user "/	
	./•	P laterpreter. */	

Diffs	node, Dirrid. Junique), Affallstate Cratefile., Affallstate. TotalCalls., 1. (1741afaseat (Mase)) (	// Oct sescisted volume/wode for the persat dir; callet Jong are also returned */ if (errorceds = Octrolime/packegitron, Oiffid, Evolptr. Sparentpir, Numteerlis, carbatespettoodir, follow, Wilty_Lock, Stights, damyrights) goto Bad_crestevile;	/* set volume synchrobisation information ./ 1137, 1133 // Get associated volume/woods for the parent dir; caller long are also measures 1 [ [dirorfods - GetVolumePackage(form, Dirfid, Boolpre, Sparentpher, 1981	_createfile; ume mynchromisation	*** ser_sex ****  *** 1116_11151 ****  *** 1116_11151 ****  *** 1116_11151 ****  Goto Bad_Creater[is.	/* Check ACP, too. */  * If (tel = NOLA) {  * Tel getystfel, **same*, Neam);  * TelDeficoefisturefisturefic), *forteru", Infertua);  * Information, **same*, **remaintel*, **forteru*);  * TelDeficoefistureficoefisturefile;  * Opto Not_Createfile;  * Opto Not_Createfile;		**** 1911,1016 ****  **** 1911,1016 ****  **** 1904ta and atore volume/wade and parent woods back */ **** Pubble and atore volume/superioritis, targetpis, parentpis, volpis);  ***********************************	19 1243,1389 1940,1389   ' rights for this user '   ' rights for any user ' rights for any user '   ' rights for any user '   ' rights for any user ' rights for any user '   ' rights for any user '   ' rights for any user ' rights for any u
	/* pointer to olimn: atrocture */ /* rights for this and any user */ /* Act iterpreser. */ /* Access control program tent */ f we need one. */	tel - Teldoriteon, datpl: Yeloogi, 1988 Resovabile te. Did - tu.Md.Mdn", Esse, Dirfid->Volume, Dirfid->V Affeciliènte.Resovabile., Affeciliènte.Aptalonie,	/* Out yoluma/Annida for the parent dir. caller's access rights are also returned if ferrorcode - GerVolumaPackegetcoo, Dirrid, Avojptr, aptromptr, dustabols, demrestabilizaryetsotdir, dustabilish armstyllar, good did_femonsfile, eet volume swithinglasion information */	1919, 1934 Af (errorCode - ChechwitteMode(parentptr, rights, Pasts Dilgits))	oh. too. •/ • Hanel j	Trenovefila", "dir", "delata")) { 18; [lb.] dosired file ./	interior of the contract of th	umavodos and parent races back */ Usbersagetocidir, tergespit, parentpir, volptr); 'ile returne (d\n', errorcode);	/* Mandie for dir package 1/0 ·/ /* Pointer to client etructure ·/ /* Lights for this and any user ·/ /* ACP interpreter. ·/ /* access Control progras text ·/ If we need one. ·/  >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
	struct ditent a citent; long rights, enyrights; Tel_Interp 'tel; char 'ecp; /* Oet en interpreter if	tcl * Tidet(tcob, &tcp): Vicelog(1, *APE_ResoveFil) Diffid-)Unique); APECall#tate.ResoveFile+,	/* Get yolume/winde for If (errorcode - Get Yolum GOCO (in d_ Remount in,	*** 489, 484 **** *** 1019, 1014 **** *** (erorcode - Chackwri	goto Bad_Mombowefile; /* Check whether ACP is oh, too. if [tel := wmil.]   701_SerVar(tol. 'nemo', Hene);	if ilacroates), esp. rr- errorode - BACOES goto Bad_Removepile; } // Actually delete the dos	15 (05707000 - Deleters 15 (05707000 - Deleters 16 (05707000 - Deleters 16 (0570700 - Delet	Tugode and store would be united by the unit	dir, mm e diient. F. acyrighte etdi: Interpreter : Ettoom, &eq

Diffs	103,1318 1 (crorrode - checkfiledtodelevrott, newrighte, Pasfe_lHSEAT)) 1 (crorrode - checkfiledtodelevrott, newrighte, Pasfe_lHSEAT)) 1 (crorrode - checkfiledtodelevrott, newrighte, Pasfe lHSEAT) 1 (crorrode - checkfiledtodelevrott, newrighte, sequence   Crorrode - checkfiledtodelevrotte, se	to deptrdisk.oloned Copyonrite(oldrott, volptr);  # if (oldroptrdisk.oloned) Copyonrite(oldrott, volptr);  ## if (oldrott, volpt	PurfolumeBackge(filabtr, (mewrptr & mewrptr !- oldrptr? mewrptr : 0), eldrptr, vo lott), ViceLogia, 'arts memer et d'ar', etrorCode);  * Palmaleme(tel);  return etrorCode;	*** 1940.1345 ***  Volume * volptr * 0.  Extruct direct * olient; /* pointer to the volume badder */  Stor fights * apyrights; /* rights for this and any uner */  **********************************	controllers and and an action and an architecturation and an architecturation and architecturation architecturation and architecturation architectur	// Ont the wrode and volume for the perent dir along with the celler; s rights to it i if (exroxcode - OstVolumeBuchage(tocm, Dirřid, svolptr, Gebreatptr, MustBaDlB, Gebreatmystrotdir, Golient, WITYB_LOCE, Arights, Anyrights) goto had_Pyminki // set volume synchromissico information //	/* Out the woods and volume for the parent dir slong with the caller's rights to it.  if (arrorcods = GetVolumeDatabage(toom, Dirlid, Broiptr, Aprematpir,  i MusiBulli, Spiritsbarperootdir,  i dollame, WAIYE_LOCE, Arights,
	. tol - toldet(tode, &aco);  volume(). "Ara_mense to to, fld - tu.Nd.Nd to tu.Nd.Nd/or, olderse, Herdans, oldsirid." Volume, Herdans, oldsirid." Volume, Herdans, Herdans, node, Manolifid. District. Volume, Herdans, Herd	<pre>ipacent, 4cilent, Matre_Lock, atighte, samyrightm)  </pre>	i if (errorCode - OstVolumbPackmysitom, NawOlrFid, Gvolptr, Mammptr, NustBaDIB, Sparsn, Golder, MRITE_LOGK, Ansverights, Saswangrights)  1 If (errorCode - OstVolumsPackmysiton, OldDirFid. Svolptr, Goldvptr, NumEBaDIR, Sparen, Goldver, Maleladia, Affekt, Sasyrights)  1 goto Mad_Maname;  1 (errorCode - OstVolumsPackmysiton)	SE (errortode - Oath (err	it (Johnston France)  Instruct of Order;  Instruction of Control and Control and Control of Control	olume Packagus (p. 66. 66. 66. 66. 66. 66. 66. 66. 66. 6	goto Bad_Ranama,    goto Bad_Ranama,

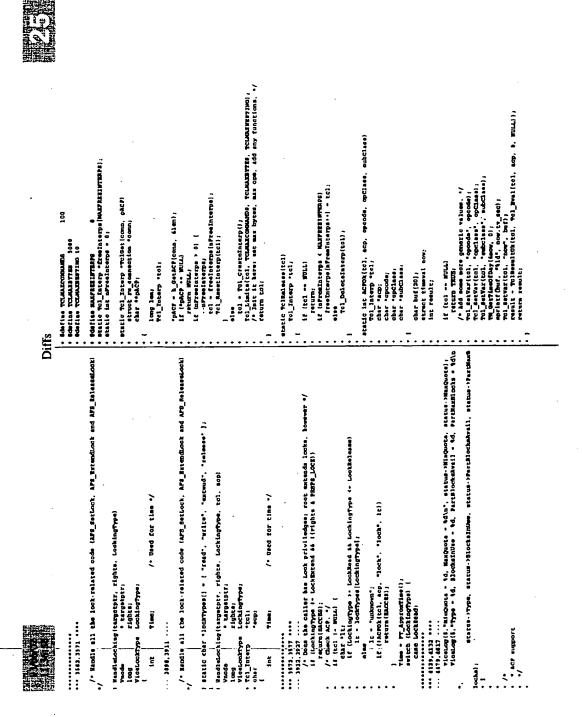


	ā	Diffs	
/* set volume eynchronistion information */ 1820, 1829	aformation */	••	Tol. Interpreter. '/ ACP interpreter. '/  (' access control program text '/  (' access control program text '/
/* Get the wands and volume for	/* Get the wacds and volume for the perent dir slong with the veiler's rights to i	•••	/* Get an interpreter if we need one/
if (errorcode - GetVoluenPeckeg	Mechapeltcon, Dirfid, dwolptr, dparmtptr, hartbedlm, dparentnimetargetmotdir, bollent, Wlyfellock, arights, debyidstr, cil)	÷	Vicelog(), "AFF_RemoveDir Ne, Did " tw.Nd.Nd\n", Mass. DirFid.>Volume, DirFid.>vno DirFid.>Unique); AFBChliffate.NesoveDir++, AFBChliftate.Totalcalis++;
table:		:	/ Get the whode and volume for the parent dir slong with the caller's rights to it
/* set volume synchronization information	formation */	· 	if (errorcode a GetVolumePachaga(teon, Dirrid, évolptr, Aparentptr,
*** 1597,1603 ****		. <u>-</u> -	Mailedunk, bassatthertargetnotdir, bollant, Whire Lock, krights,
if (errorDode - CheckWritcModelp goto Bad_MakeDir;	if (erroroads - CheckWritcModelparentptr, rights, PR076_INSEXT)) goto Rad_Mabol:		gota Bad_kemoveatr;
· /* Chent Act. •/			/* set volume synchronization information */
· if (tri i- MULL) ( . TG) derVar(tG), "mene", Mene);		:	1700,1700
TelDefstorsStatus (rcl. "Instatu	Anathreas, Industra),	: :	is (arrorCode - CheckWriteMende(parentptr, righte, PREFE DELETE))
* 1F (!ACFOR(Ed), ecp. "makesir", "dir', "ineert"))     efforcede = EACCES!	', "dir', "ineert")) [		goto Bad BesoveDir;
goto Bad Katable;		٠	Check ACP. •/
			16 (ccl le mult.) ( Tol Bardas (col mana)
. Marine motivates			if (iacpokited), acp, "removedir", "dir", "delote") ;
/* gat a new whode and set it up */		· ·	errorcode - MACCES, moto Tad Democation
<pre>1f (errorcode = Alloc_Mownode(pu</pre>	1f (errorcode = Alloc Mowthode(parentpir, Sparentdir, volptr, Stargotpir, Mase, Ou volgeotory, Experiblabiocks);		
****************		•	
			/* Do the actual delate of the desired (empty) directory, Mane */ 18 (errorcode * Deletefarget(parentht; voloty, hancery any sair sairs.
Purite the all modified vacdes	vacdes (parest, Any files) and volume back ./	ust Bedik)	N '0000 '010000'0 '0100' '010' '010' '010' '010' '010' '010' '010' '010' '010' '010' '010' '010' '010' '010
Makedir r	Natural arrandode);	:	Coto Med Removable;
relations of the relati		: :	*** 1719, 1726 **** *** *** *** *** *** *** *** ***
-		:	/* Write the all modified whodes (parent, new files) and wolume back ./
*******			FULVOLUESTERAGE(PERENTRUDORIENTSCHOOLGER, (ARGECOLF, DEXEMIDIT, VOIDER);
1485, 1485 Volume - volptr - 0;	/* pointer to the volume header */		Tolkalase(tol); return errorcode:
struct oliemt * olient; long rights, enyrights;	/* pointer to elient structure */ /* rights for this and any user */	-	
Viceing(1, "AFF ResoveDir te, Di	Did - tu.td.td.a., Bame, Direid-volume, Direid-ven		*** 1762, 1767
AFBCallState. RemoveDire.	AFSCallState, TotalCalls;	:	3037,3034 etruct client;
/* Get the whode and volume for	is for the parent dir along with the maller's rights to i		ktype[3] -
1 12 (errordode - detVolumePackage	Package(toom, Dimfid, evolptr, sparentptr, Rustlenik, apa		Tel_Interp *edl; /* Acr interpreter. */
rentubentergetnotdir, kollant, Whire Lock, Arights, Sacyrights); goto Jed_Resovebir;	OCK, trights, tsoyrights);		/ stress control broaklesd 64 twos is Londerites; s
- sysobroal se	tion information o/		errorcode « EIWAL,
1912, 1951		*** 17	*** 1771,1778 ****
etruce client of and long rights, anything	/* polities to the volume header */ /* polaties to the volume header */ /* polaties to this and any mean */		locktype[(intltype], Fid-)Volume, Fid-)Volume, Fid-)Volume, Fid-)Unique], Afficiliftetm.SetLock++, Afficiliftete,TotalCalle++;



		Q	Diffs
It involper, Stargetper, Domergher, Garea	" Get the wode and	the desired file along with the caller's rights to	goto Bed Getvolumestatus;
demayrid, Tollians - avoild,  olises to the volume hander -/ colises for this and any user -/ colises for this and any this the oblise's rights to collises the colises for this and any tist the oblise's rights to collises the colises to collises the collises coll	if (errorceds = detvolumePathrge twhentergetrockir, Anire, Whire Lock goto Bad_ReleaseLock;	(tcon. Pid. Swolptr. Stargetptr, DOSFCIECE, Sparen., Srights, Sanyrights)	12 ([Manillavior(clicati) && (!(rights & Plabs_Read))) (
ointer to the volume hander // interference of the volume and any uner // interference of the volume and of the volume and the vo	/ eet volume synchronization has		duamyPid.Tolume = avolid, GamyFid.Vnode = (long)BOOTVHOOE, dummyFid.Unique - 1;
Good Bad_Batters   A	Volume * volper * 8; etruct cident * cilent;	/* pointer to the volume header */	1 1f (errorcode - OetVolume Package (tcco. admanyPid, avaiptr, atargatptr.
		/* Iights for this and any user */ /* ACP interpreter. */ /* ACP interpreter. */	_
*** **********************************		*u.1d.4d\m", Fid.>Volume, Pid.>Vonde, Pid->Vonique	if ((Vanilladwar(clienti) && ((frights & prefg_NRAD))) {
** ** ** ** ** ** ** ** ** ** ** ** **		allatets.TotalCalla,	
** ** ** ** ** ** ** ** ** ** ** ** **	. for det an interpreter if en need to tel e TelGekitton, sachi:	Dne. •/	
This is not been a second and a second and a second a sec	•	the desired file slong with the caller's rights to	Tol Interp +tol;
triples (CCE, drights)  (co. 1)  (co. 1	if terrordeds - detVolumePachag	trom. Pld. brolptr. Atargetptr. Marcatch, Sarrestanternetari	ViceLog(1, "AFS_GetVolumeStatus for volume Auln", evolid);
if (Geroscode - Californamistates)  - /- One as information in the control of the		illent, warte Lock, drights, toyrights, cci);	*** 30.0 23.0
* /* One as interpreter if we need one. */  * * * * * * * * * * * * * * * * * *	/* set volume synchronization inf	formation ./	if (erroscode - callFreemble(struct ra_call **) sicon, acrrvecala)) goto Bad_detvolumestatus;
APGCALISTATE GECVOlumeStatus., APSCALISTATE, TOTAL CONTROLLES TOTALISMENT.  If (avoid — 0) (  ***********************************	*** 1907, 1913 **** SetVolumeSync (6ync, volptr);		10 P
rich by triggered here */  rich by triggered here */  richts, LockRelease, tcl. acp)  biscutchering conditions a void dummyfid. Whode * (Local Edothucia)  if (stroctode = detVolume*status)  biscutchering conditions and detVolume*status)  if (stroctode = detVolu	/* Mandle the actual lock release	/ getptr. righte, LockReleagi)	
demay71d, Volume - avolid, dumayFid. Unde - (1009)EDOTVNDDE.  If (errorded derVolume*status)  If (errorded derVolume*status)  If ((vanilade. et/s) as ((rights 6 propertion))) (	goto Bad_heleasetLock; /* if no more locks left, a calif-	And because of Lines of	
if (errorcode - dervolumestatege(con, demanyeld, stopper, parentubantaryerocdir, acident, anaptege(con, demanyeld, stopper, basyriphte))  outd be triggured here */  if (errorcode - dervolumestates)  if (arrorcode - dervolumestates)  if (arrorco	2158, 2306 EstVolumeSync(Sync, volptr);		dammy Fld. Volume - avolid. dumayeld. Vnode - firms severans
ould be triggered here */  r. targetper, (fmode *!), volptz;;  in', stroroods;;  se * (long) moorferms, demayrid.cuique * ';  sedumgrid, evolptr, tergetper, masthaniz, 3s  sedumgrid, evolptr, tergetper, masthaniz, 3s	/* Mandle the actual lock release if (errortode - MandleLocking(tar goto Bed_ReleaseLock;	/ :getptr, rights, LockRelesse, tcl. scpl)	1 1f (errorCode - detVolume)achage(tcon, adummy#1d, avolptr, tiargetptr, bustBeDIR, parentwhentargetpocdir, aclient, aka_Lock, arights, hayzighte)] gote 84d_GetVolumeStatus,
r. tarpstptr, (Woode *)0, volptr);  in. strorcodes;  sections) Moorwands, damayeld.Calque * 1;  sedamayeld, evolute, damayeld.Calque * 1;  sedamayeld, evolute. desseptptr, Mosteballe,	/* if no more looks laft, a uslib	sack would be triggered here */	if ((VasillaDeer(client)) as ((fights & PRSP8_TEAD))) (
r, targetper, (Woode *10, volpts);  strorcode);  section() norworms, demay*id.Caique - 3;  sedammy*id. dvolpts, desgetper, NostEadle,  ***********************************			demyPid Tolume - avoild dummy did Vande - front sootenme
de * (long) moorwunk, deamyfid.baique * 1;  de * (long anoorwunk,	/* Put back file's whode and wolm PutVolumePackage(parentwhentarget Vicelog(8, A/Pa_kalesselock return		1 if (errordode - GetVolume)valangeten, totampfid, twolptr, ttangetptr,
ss - (long) moorwans, dammyrid.Chique - 1; 3s dammyrid, dvalue c. 1; 3s dammyrid, dvalue c. 1; 3s dammyrid, dvalue c. 6:1; 3s dammyrid.	<ul> <li>folkoleaes(tol);</li> <li>return(errorCode);</li> </ul>		_
de « (long) mborranne, dasayrid, mattenin, 1803. de damayrid, avojet, designopt, mattenin, deichte.	*** 2264, 2370 ****		if ((Tunillauberiellenel) 44 (ffrights 4 Phare_main)) (
ddummyfid, Swolptr, Stargetptr, Musthellin,	i dummyFld.Volume * avolid, dummyFl	d. Vhoda . (long) mortwunk, daamyFid. Daique - 1;	*** 2320, 2324 **** *** 2423, 4627 ****
Construction to the construction of the constr	i if (errorCode - GetVolumePackige(tcon, Aparentylenterretric) and LOCE,	tcon, adummyfid, svojptr, stargetptr, musthabir, . [OCK, drights, sanyrights]]	<pre>goto Bad GatVolumeStatus; }</pre>

	/ Chart Act. /	SIIO
- •		(ICM With the service of the service
•	TOLDELATICAL, "volume", avolid); if (jacroh(tol, scp. "getrolumestatus", sterch" manimus;	1621, 2631
	Groroda - RACCES; Goto Bad datVolumeEtatus;	/* This unusual long-parameter routine encapsulates all volume package related operation on together in a slonds medials from related by since the second of the slonds and second operation.
	1	/- willed the interface only -/
•		o de l'outransmedinge (com. 714, volptr, targathtr, abhforolt, parent, client, lookippe, rig hte, asyrighte)
	BUCCALVolusesCarus (Perchvolstetus, Mane, Offilmeneg, Motd. volptr);	Struct Foomsection stoop / R-consection */
	Btatus	** ** ** ** ** ** ** ** ** ** ** ** **
•	***************************************	Pid 4/
•		
	td == 0) (*Motd . (char .) sallog(1): sewest = 0.1	/* this unusual long-parameter Fouries encounter-
	malloc(1), "Offitsaling . 0.1	ne together is a single module, it a called by alsons all as the perfect only a
•		GetVolumePedhage(tgom, Pid, wo)nry remarkes abtoness
		locktype, rights, sevrights, collocal, parent, dident,
	_	Struct Counscilon teen; / R.Connection ./
•		ovology: / Pid that we are
•	*** 2306, 2393 ***	
		*** 2637, 2613 ***
•	1.1 ang 1 katawa danay 1.1 ang 1 ang 1.1 ang 1	"parent; /* If 71d not a dir.
- 4	If farranced = GetvolumePackage(tecm, adumnyPid, evolptr, atargetotr, MustBebis.	
l	_	"rights, "seyrights, /" Returns user; a A and and all and a
•	2690,2100	ing activities (412, /* Internal Access (102 */
	designation and the state of th	t clientclient,
	if (errorcode - GatVolumnPachage(tcun, admmyFld, avolptr, atargotptr,	•
-	601fert Read Local Addit.	
-		TC1_loterp *tc1, /* if non-Wull. set Some variables have a/
	goto Bad Bervol westerns	The second secon
	leCher(client)) (	int actions of the control of the co
•	2465, 2451,	oto Macca maga to the state of
		・・・・ 登画は
		(*cigbte) &* -PRSPS_ADMINISTER;
_ \$	Adamsylid, evolptr, Atargotptr, Masthebis.	
•		=
:	18 (VanillaGest(client)) (	TOT BELOW !
		A LEVERING - CASE LIVE - FILE - FILE - (*EASTGETTE - CASE LIVE - FILE - CASE -
	dammyFid.Volume = avolid, dammyFid.Vaode = (loug)BOOTVBOOB, dammyFid.Unique = 1:	
-		Action of the Park
-	" Ustvoistanterage(toom, &dummyPld, &volptr, &targetptr, MostBeD12, Aparentalmonearations.	(*parent) - ywodekumber, (*parent) - yd (* inferiela)
	bollent, RMO_LOCK, arights.	
•	goto had_derWolumestatus, tanyrights, gota had_derWolumestatus,	TelDefüriditel, "parentfid", 0, 0, 0)
	•	
	ir (VanilleDser(client)) (	return errortode;
	-	
	-	



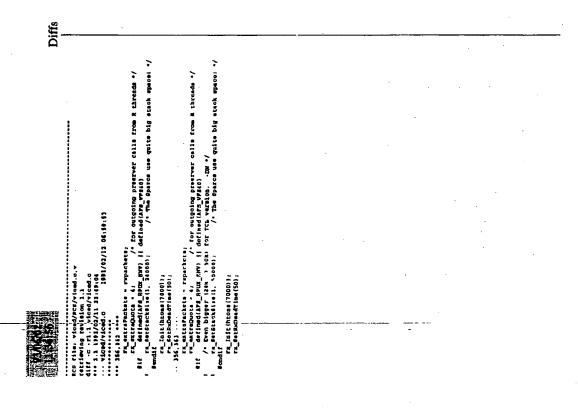


Diffs

The agents with a hiceger value. \*/
The agents with a hick with a hick









## **XFROX**

## nichols

Document name:

**TCLDiffs** 

Printing date/time:

June 2, 1993 11:46:57 am PDT

Host name:

osprey

This file, "TCLDiffs," represents the differences in source code between release 3.3 of Tool Command Language and a version of Tool Command Language modified from release 3.3 to support the embodiment of the present invention in a modified version of the Andrew File System as described elsewhere in this application.

The modifications made to release 3.3 of Tool Command Language in order to produce the embodiment of the present invention in a modified version of the Andrew File System as described elsewhere in this application represent original unpublished work of Xerox Corporation. This original unpublished work is Copyright \*1991 Xerox Corporation. All rights reserved. Copyright protection claimed includes all forms and matters of copyrightable material and information now allowed by statutory or judicial law or hereafter granted, including without limitation, material generated from the software programs which are displayed on the screen such as icons, screen display looks, etc.

nichols

(120.75) (127.75) (127.75)

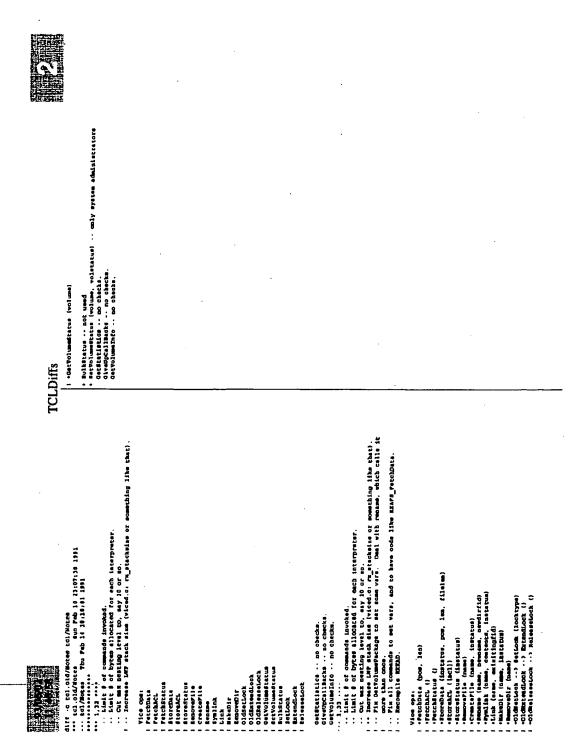
C05d: Mon May 24 12:24:03 PDT 1993

Job #589



**TCLDiffs** 

```
diff of Colodifinatells of Colombias (1) of Colodifinatells of Colodifinatells of Colodifinatells of Colodifinatells of Colombias (1) of Colom
```





X 00 1



**TCLDiffs** 

### Condition of the co

100,199

130.336

Taging ther = (Interp \*) interp.

Interp \*) interp.

I (oftr = (interpallach \*) mailor(sized(interpallach));

(oftr-)sire \* pro:

10.513

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

10.515

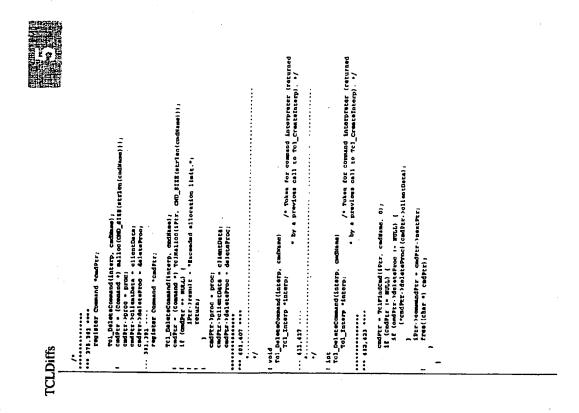
10.515

10.515

10.515

10.515

10.

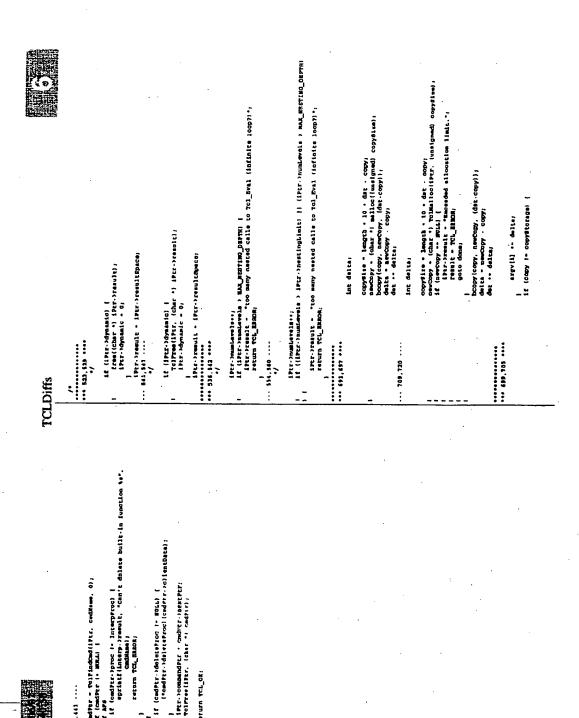


/\*
101,332 .... idelterProc i\* BULL) [
['comfre.)delterProc) (confre.)cilestate);
['comfre.)delterProc) (confre.)cilestate);

) folfrem(iftr, (char \*) cmdftr);

white (tets-)revers i- MILL) (
TCleres(fets, (char +) (pts-)revets);
APCs-)revets - tets brouper-inemepter; If (Sper. Sevence to MULL) ( TCIFram(19tr. ) events) ;

01/09/2003, EAST Version: 1.03.0002



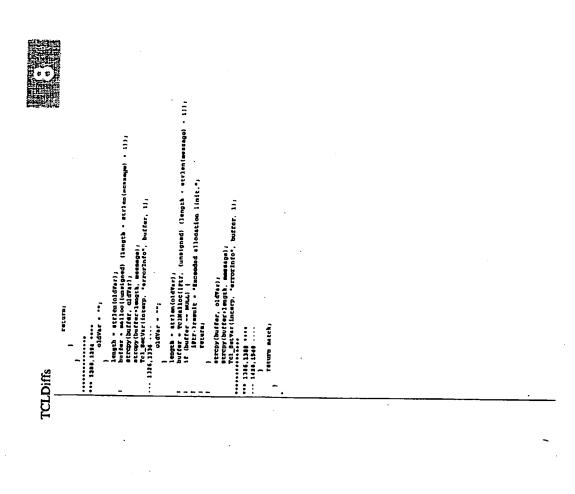
omdfur - folfindomd(lftr, omdMame, 0); if (omdfur i- mm.k) i

.... 432,443 ....

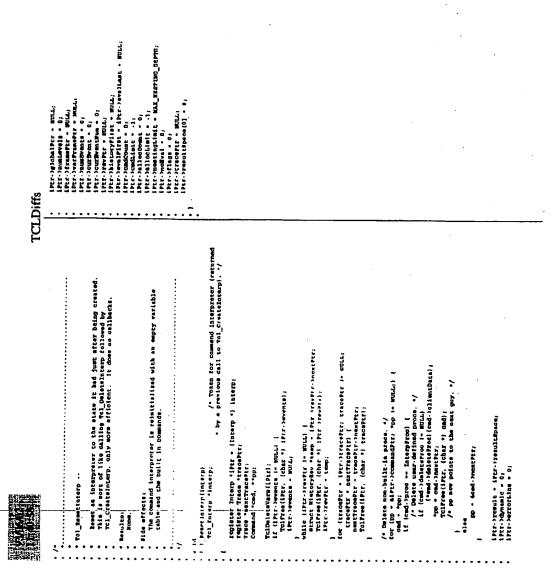
istr.)compander · cadher.beater; folytee(itt. (char · cadher);

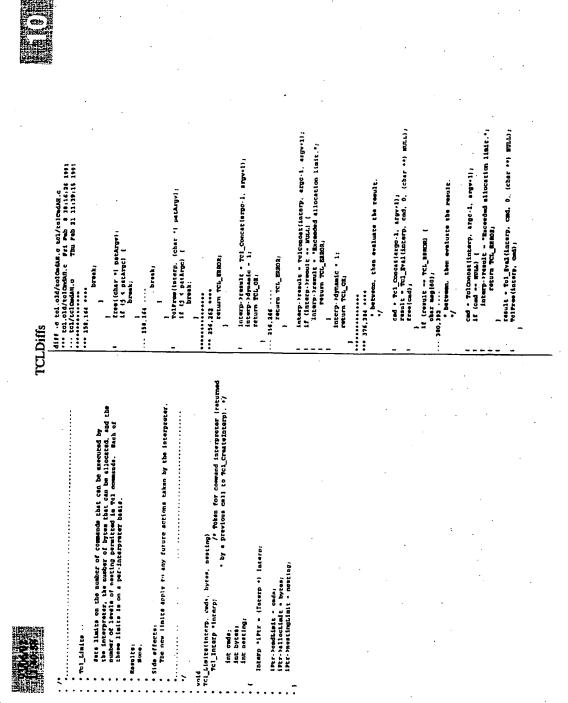
rerurn TCL\_OK:

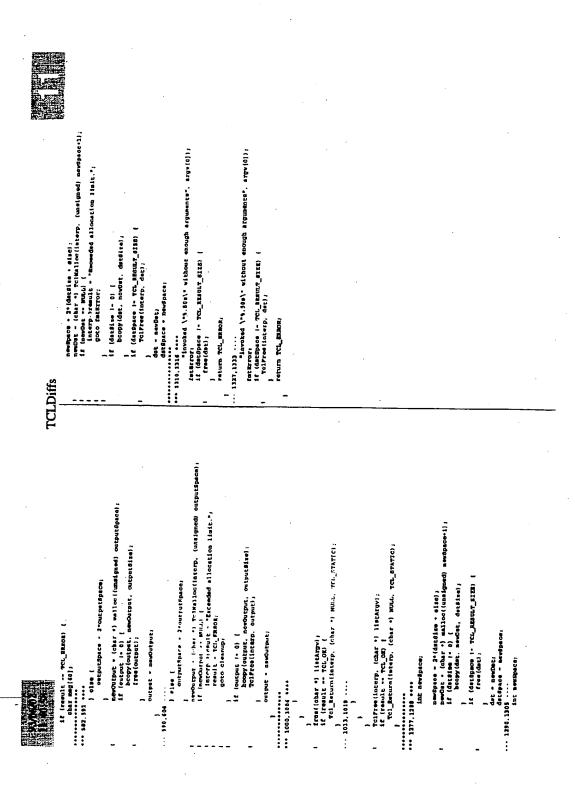
							_							_				
Diffs	int delte,	<pre>copysise == i;</pre>	det - delte; for (1 = 0, 1 < expp; 1 · · ) ( expv(1) - delte;	i (apy :- apydeauge) ( i frac(chr *) apyl ;	dopy = newCopy; 1 list: newCopy + copysise . GUFFER; 11 (nr Andres)	ooyystas - 3; newCooy - (char e) whitehington	if traccopy - NOLL   intrapped octypits)   if traccopy - NOLL   intraccopy -	goto does;    popy (copy, exectory, (det.copy));   delta emedone.	(eqipp (gi/abs - app	if (copy is copyStorings) ( TolFree(lFt, (chet *) copy); copy = bescapy)	limit - newdopy - copysiseurrex; 991,959		free(Coar 1) fre-yrault ,   ftt-ydymanc	*/ 1Ptr->cadcomt**; 1Pt	result - red_ERROR; gets done;	iPtr-Pings 6 TOO, IN PROCESS; if (Aftr-Obrasio) ( TOO: TOO: TOO: (Ohr O Aftr-Obrasio	SPEC->teault - SPEC->teault@page;	*** 1010,1019 ****
TCLDiffs	free((dhar *) copy);	copy = newCopy; limit = newCopy = copySize - BUPPER; argv[1] -= delta;	if (copy to copystorage) ( Telfreelibts, (char *) copy),	COPY * secCopy; Link: * newCopy * copyelie . UUFFER;	gate done;	Aptroductor of Percentage and Aptroductor of Aptrophysics of Aptroductor of Aptrophysics of Aptro	doto dores	if (leg: obcasic) (	irtr.sreeuic - value;	arplite ** 1.  newloge * (char **)  newlog (char **)  for (1 * n * ( char **) )	-	argv - newkina;	argeise 2; newargo - (char) TClMalloc(iPer, (unsigned) argeise - except (char -))	if (newargs MULA)	for (4 - 0; 4 < argc; 4*) ( newArge(4) - ergw[1];	if (argy is argatorage) ( volves(ifut, (char *) argy);	indivant Are	3
	-	725,728	-	***************************************	-		749,755	_	*** 80¢			627, 643				-		

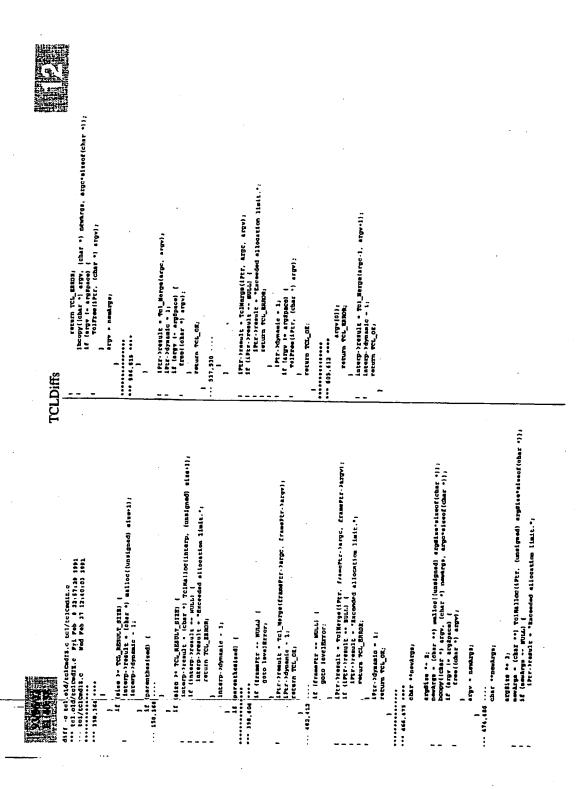












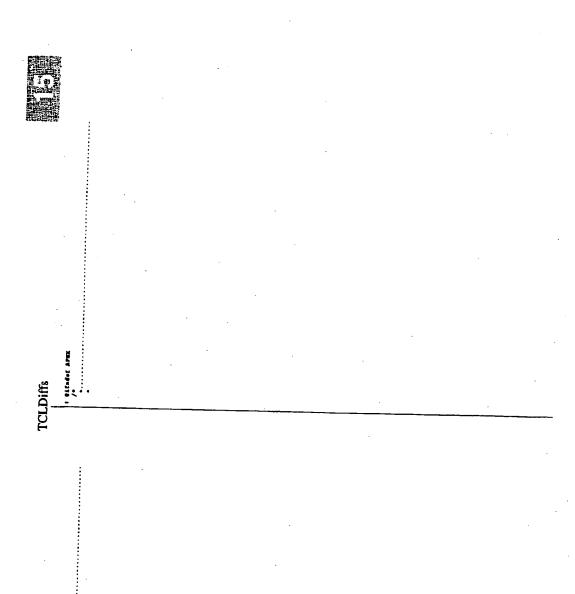


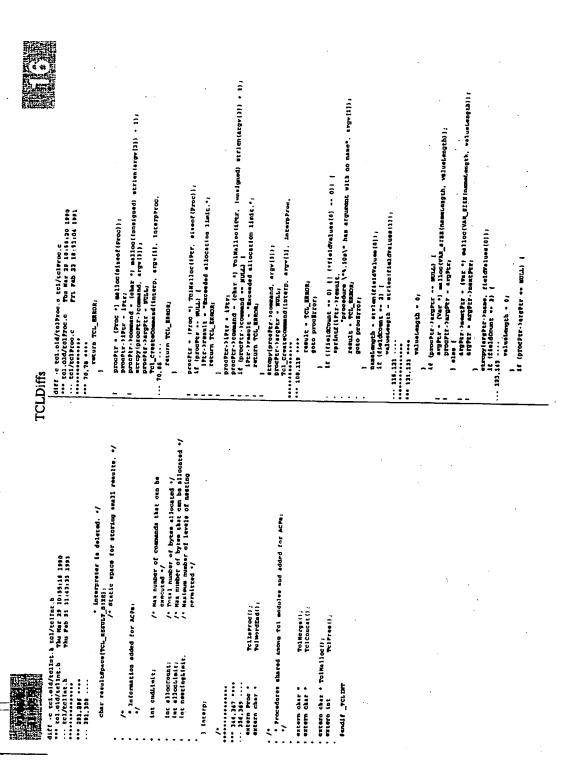
and the state of t

01/09/2003, EAST Version: 1.03.0002

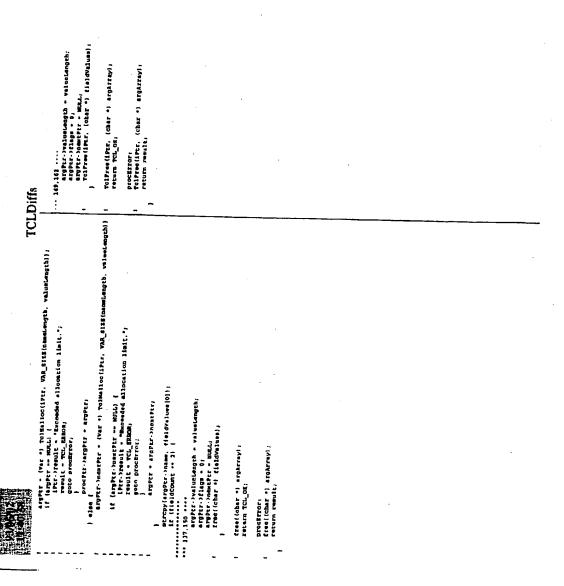


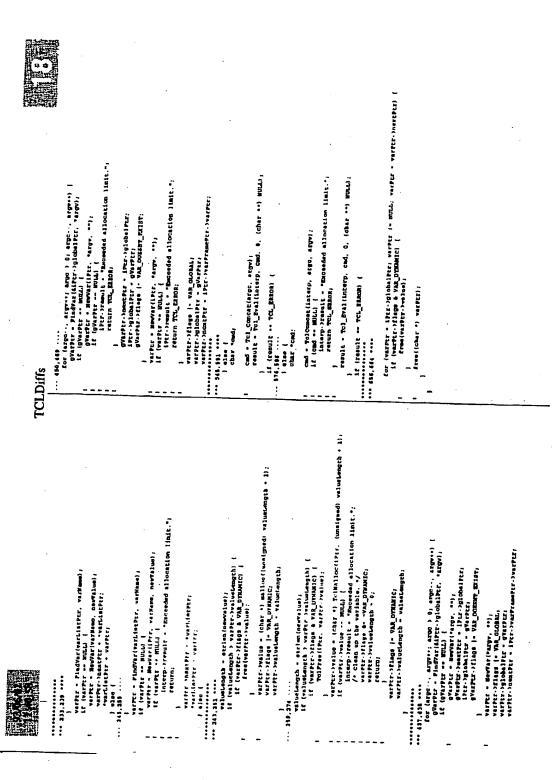
I Elas Carlos de	CLDITIS
· ADVENTED AT	Tel_AddBroringo (antere, asg);
	Tollree (interp. cadhaffar);
formation (court of Tolkalloc(interp, (unsigned) totalsise);	Zeturn result;
latery-)result " "Exceeded allocation limit.";	Wendl ?
	*** 1372, 1278
for (1 · 0, ketaläise · 0, curpield · fields)	return fcL ok
Curple3d-)location - washing	-
total fits . curtical . size;	
500000000000000000000000000000000000000	
. Attack	
reter 141, 48	-
1133 1336	
break	-
	_
SONTONE CHARACTER STORY TO THE STORY	
return TCL_OK;	
	<u>-</u>
** 1162,1169 ****	
dlose(file1d);	
return TCL_BAROR,	
cudbuffer - (char *) malloc( una mad)	
if (readisized, cudmiffer, lint) Statubuf.or mise) to Statuming also)	
olose(filleld), "error in reading file (*8.50s)", argv[1]);	
return TCL_ERROR;	
Caddauffer - (char e) toward actions	
if (cadmiffer - MRL) (	
interp. Natural . "Encomeded allocation limit.";	
if (read(fileId, cadBuffer, (int) statBuf.et_size)   statBuf.et_size)	
olose (fileId),	
return FCL_EDBOR;	
** 1184,1190 ***	
Optint (and, " (tile \". 50s\" line td)", argv(1), interp. lessorites);	
free(undaufer); return result;	
-	
# endis#	-
Sprints (sag. * (2134 \*0.50s\* 110s 4d)", arevit) december 1	
Commission and the commission of the commission	

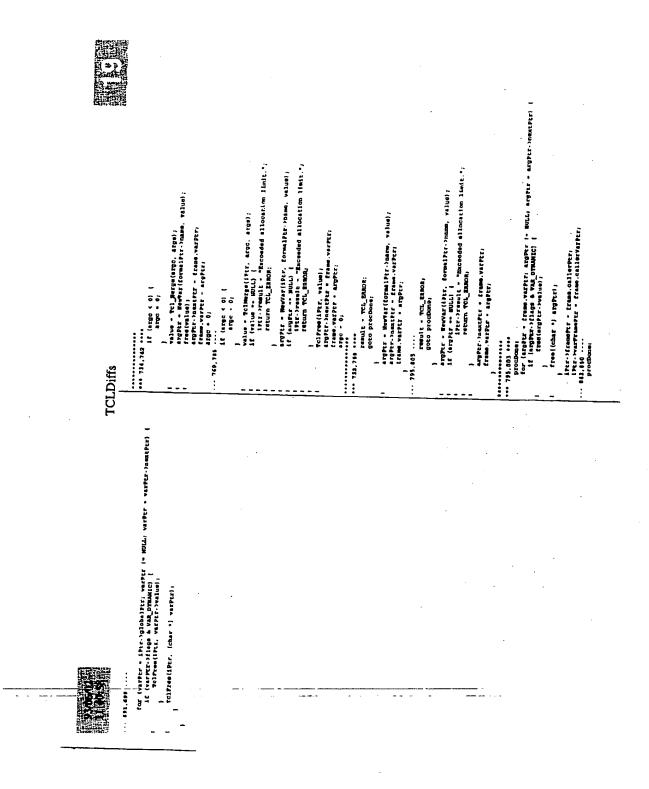


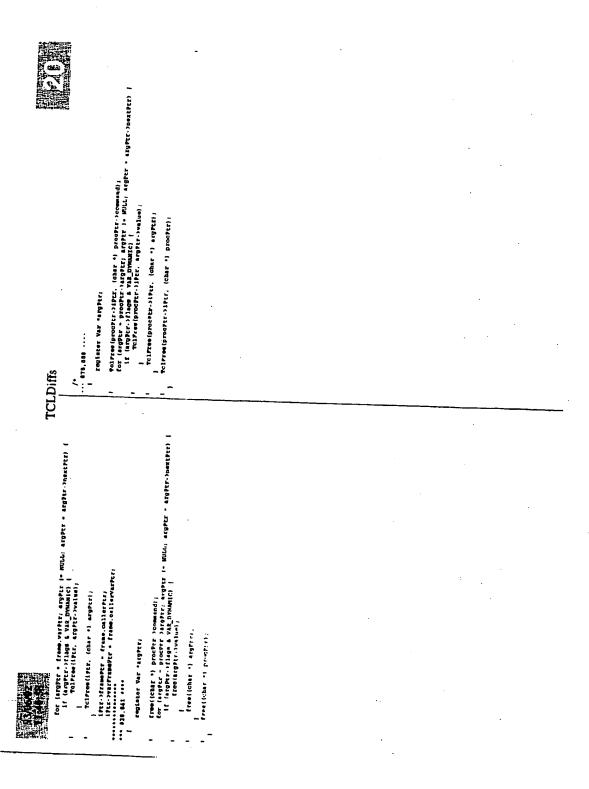


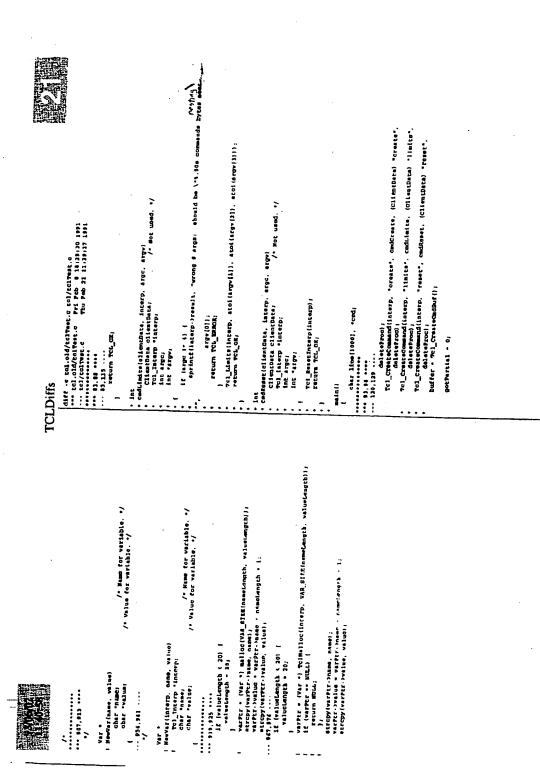


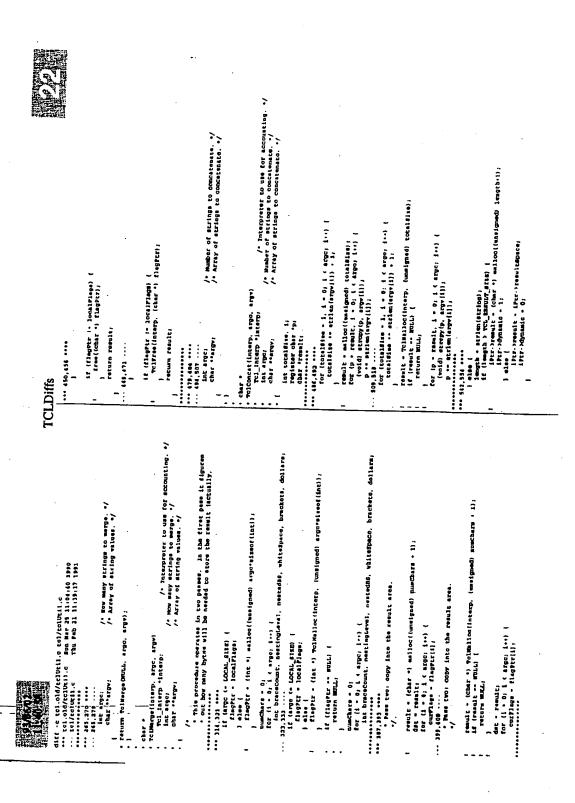




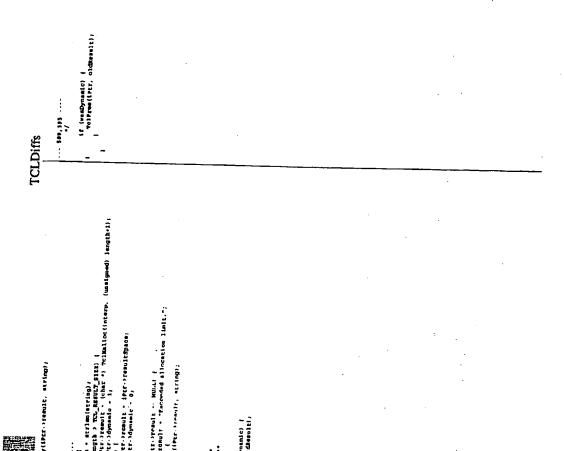


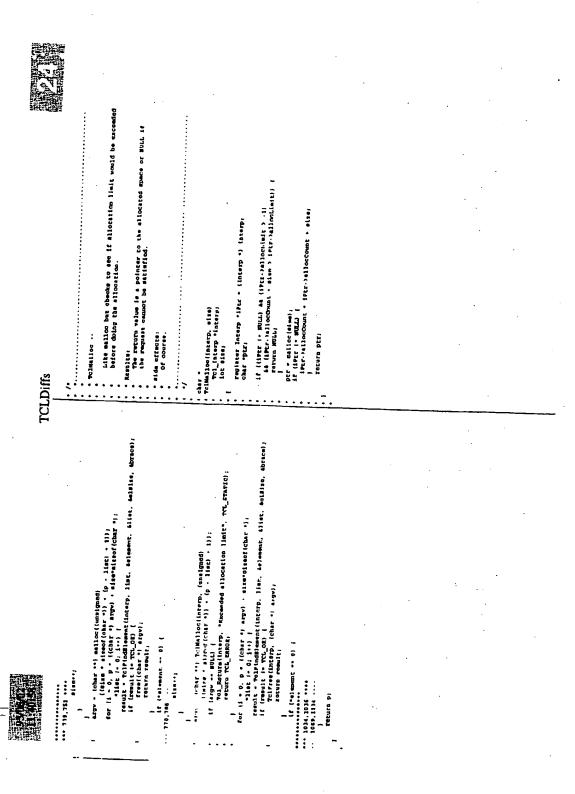














<sup>18</sup>/071649

TOLDIES

Tolored to the second of atomical to free and the second of atomical to free and the second of atomical to free and transfer and the allocation limit with the second of atomical to free and the second of the second of

What is claimed is:

1. In a computing system comprising a server, a client, and at least one intermediary, a method of processing an ultimate request to the server, the ultimate request being delivered to the server as the final request in a chain comprising at least two linked requests, the client and all the intermediaries each being associated with one linked request of the chain, the intermediary that delivers the ultimate request to the server being the final intermediary in the chain and being designated as the requestor, the method comprising the steps of:

using the requestor to present to the server the ultimate request in conjunction with at least one executable access control program comprising at least one sequence of computer program instructions, the access control program being executable by a processor to 15 express a specification of a set of access rights;

using the server to execute each access control program thus presented, each access control program being executed in a manner such that said access control program is prevented from compromising server security; and

if and only if the execution of each access control program thus presented is successful, using the server to execute the ultimate request in a manner consistent with the set of access rights, any access rights not in the set of access rights not being delegated to any intermediary nor being granted by the server.

2. In a system comprising a client, a server, and one or more intermediaries that the client does not trust, a plurality of communications channels that connect the client, the server, and the intermediaries, one or more computing nodes, and a communications network, a method for performing a request issued by an intermediary to the server on behalf of the client, the method comprising the steps of:

using the client to create a client request;

using the client to create an executable access control program, the access control program comprising at least one sequence of computer program instructions, the access control program being executable by a processor to express a specification of an arbitrary set of access rights to be delegated from the client to a first intermediary untrusted by the client, any access rights not in the set not being delegated to the first intermediary;

using the client to associate the access control program with the client request;

in response to the client request, using the first intermediary and zero or more additional intermediaries untrusted by the client to issue intermediary requests, all these intermediary requests being accompanied by the access control program;

receiving a final intermediary request in the server; and using the server to execute the access control program in order to determine whether or not to grant the final 55 intermediary request, the access control program being executed in a manner such that the access control program is prevented from compromising server security.

3. In a system comprising a client, a server, a number of 60 intermediaries, the number being greater than or equal to one, a plurality of communications channels that connect the client, the server, and the intermediaries, and computing hardware to execute the client, server, and intermediaries and to support the communication channels, a method for 65 performing a client request issued by the client to a first intermediary, the method comprising the steps of:

using the client to create the client request;

using the client to create an executable access control program comprising at least one sequence of computer program instructions executable by a processor, the access control program being executable to express a specification of a set of access rights to be delegated from the client to the first intermediary, any access rights not in the set not being delegated to the first intermediary;

using the client to associate the access control program with the client request;

using the client and a communications channel from the client to the first intermediary to transmit the client request and its associated access control program from the client to the first intermediary;

using the first intermediary to generate a first intermediary request;

using the first intermediary and a communications channel to issue the first intermediary request and to transmit the access control program along with the first intermediary request thus issued;

using the server to receive a service request and the access control program;

using the server to make a determination whether the client approves the service request by performing a test that comprises the steps of:

using the server to execute the access control program, the access control program being executed in a manner such that the access control program is prevented from compromising server security; and

using the server to check a value returned by the access control program thus executed; and

if and only if the determination thus made by the server is that the client approves the service request, using the server to execute the service request, and otherwise using the server to deny the service request.

4. The method of claim 1 wherein the number of intermediaries is exactly one, wherein the first intermediary request and the service request are one and the same, wherein the communications channel used by the first intermediary to issue the first intermediary request connects the first intermediary directly with the server, such that when the first intermediary issues the first intermediate request, the first intermediate request is transmitted directly to the server.

5. The method of claim 3 wherein the access control program is a procedure that takes as parameters a caller, an operation being performed, and any arguments required for this operation, and that returns an indication of whether or not the operation is allowed.

6. The method of claim 3 additionally comprising the step of using the client to digitally sign the access control program with a digital signature associated with the client, and wherein the step of using the server to make a determination whether the client approves the service request further comprises an additional test of using the server to verify that the access control program bears a digital signature that is authentic and that is the client's.

7. The method of claim 3 wherein the step of creating an 3. In a system comprising a client, a server, a number of 60 access control program is performed independently of the termediaries, the number being greater than or equal to server.

8. The method of claim 3 wherein the step of creating an access control program is performed at run time contemporaneously with the step of creating the client request.

9. The method of claim 1 wherein the number of intermediaries is strictly greater than one, wherein the intermediaries are ordered in an order from first to last, and:

a) wherein the step of using the first intermediary and a communications channel to issue the first intermediary request and to transmit the access control program along with the first intermediary request thus issued further comprises the additional steps of:

for each intermediary except the last:

using the intermediary to choose a next intermediary; using the intermediary to create an additional intermediary request;

able access control program comprising at least one additional sequence of computer program instructions executable by a processor, the additional access control program being executable to express a specification of a set of access rights to 15 control program is an interpreted program. be delegated from the intermediary to the next intermediary, any access rights not in said set not being delegated to the next intermediary;

using the intermediary to associate the additional access control program with the additional inter- 20 mediary request; and

using the intermediary and a communications channel from the intermediary to the next intermediary to transmit the access control program, the intermediary request and its associated additional 25 access control program, and all additional access control programs already created by other intermediaries, from the intermediary to the next intermediary:

b) wherein the step of using the server to receive a service 30 request and the access control program further comprises the steps of:

using the last intermediary to generate the service request: and

using the last intermediary to issue the service request 35 to the server and to transmit the access control program and all additional access control programs created by other intermediaries to the server; and

c) wherein the step of using the server to make a determination whether the client approves the service 40 request further comprises using the server to make additional determinations whether all intermediaries except the last approve the service request by performing additional tests, one additional test for each intermediary except the last, each additional test comprising 45

using the server to execute the additional access control program associated with the intermediary, said additional access control program being executed in a manner such that said additional access control pro- 50 gram is prevented from compromising server secu-

using the server to check a value returned by the additional access control program thus executed.

10. The method of claim 9 additionally comprising the 55 step, executed for each intermediary except the last, of using the intermediary to digitally sign the additional access control program with a digital signature associated with the intermediary, and wherein each additional test performed during the step of using the server to make additional 60 determinations whether all intermediaries except the last approve the service request further comprises the step of using the server to verify that the additional access control programs bear digital signatures that are authentic and that are the intermediaries'.

11. The method of claim 9 further comprising the step, performed for each intermediary except the last, of using the intermediary to designate an additional revocation object associated with a right to be delegated that is among the rights of the set of access rights of the specification of the additional access control program.

12. The method of claim 3 wherein the access control program is a computer program written in a programming

language.

13. The method of claim 12 wherein the programming language is a Turing-complete language and wherein the using the intermediary to create an additional execut- 10 access control program provides a functionality not providable by an access control program written in a language that is not Turing-complete.

> 14. The method of claim 13 wherein the programming language is an interpreted language and wherein the access

15. The method of claim 12 in which the programming language is an extension language and the server provides a core set of access control concepts, and wherein the access control program extends the core set of access control concepts provided by the server.

16. The method of claim 12 wherein the programming language includes control constructs and the server has state information that the server makes available to the access control program, and wherein the access control program includes multiple possible paths of execution and thereby provides a conditional access right that is conditioned on the state information thus made available.

17. The method of claim 3 further comprising the step, performed by a programmer, of using a template to specify portions of the access control program to be created by the

18. The method of claim 3 wherein the server is chosen from the group a file server, a database server, a print server, an input/output server, or a compute server.

19. The method of claim 3 wherein the client is a user command process associated with a user, the first intermediary is an untrusted program, and the server is an operating system in the context of which the user command process executes and the untrusted program is to be executed.

20. The method of claim 3 wherein the at least one sequence of computer program instructions is a sequence of computer program instructions executable by a processor after conversion into a machine-executable form, and further comprising the step of converting the sequence of computer program instructions into the machine-executable form.

21. The method of claim 20 wherein the converting step comprises interpreting the sequence of computer program instructions with a safe interpreter.

22. The method of claim 20 wherein the at least one sequence of computer program instructions is a sequence of computer program instructions written in an interpreted programming language, and wherein the converting step comprises interpreting the sequence of computer program instructions.

23. The method of claim 3 wherein the at least one sequence of computer program instructions is a sequence of computer program instructions directly executable by a processor.

24. The method of claim 3 wherein the step of using the server to execute the access control program in a manner such that the access control program is prevented from compromising server security comprises accessing server resources with the access control program only in a manner in which the client would be authorized to access the server.

25. The method of claim 3 wherein the step of using the server to execute the access control program in a manner such that the access control program is prevented from compromising server security comprises invoking with the access control program only server system operation having no undesired side effects, regardless of whether the server grants or denies access.

26. In a system comprising a client, a server, a number of intermediaries, the number being greater than or equal to one, a plurality of communications channels that connect the client, the server, and the intermediaries, and computing hardware to execute the client, server, and intermediaries and to support the communication channels, a method for performing a client request issued by the client to a first intermediary, the method comprising the steps of:

using the client to create the client request;

using the client to create an executable access control program, the access control program encoding a specification of a set of access rights to be delegated from the client to the first intermediary;

using the client to digitally sign the access control program with a digital signature associated with the client; using the client to associate the access control program <sup>20</sup> with the client request;

using the client and a communications channel from the client to the first intermediary to transmit the client request and its associated access control program from the client to the first intermediary;

using the first intermediary to generate a first intermediary request:

using the first intermediary and a communications channel to issue the first intermediary request and to transmit the access control program along with the first intermediary request thus issued;

using the server to receive a service request and the access control program;

using the server to make a determination whether the client approves the service request by performing a test that comprises the steps of

using the server to execute the access control program, and

using the server to check a value returned by the access 40 control program thus executed,

performing an additional test comprising the step of using the server to verify that the access control program bears a

digital signature that is authentic and that is the client's, 45 and performing two further additional tests of

using the server to verify the identity of the first intermediary, and

using the server to verify that the client has the rights that it purports to delegate via the access control 50 program; and

if and only if the determination thus made by the server is that the client approves the service request, using the server to execute the service request, and otherwise using the server to deny the service request.

27. In a system comprising a client, a server, a number of intermediaries, the number being greater than or equal to one, a plurality of communications channels that connect the client, the server, and the intermediaries, and computing hardware to execute the client, server, and intermediaries 60 and to support the communication channels, and additionally comprising an authentication server, an additional communications channel between the authentication server and the client, and computing hardware to execute the authentication server and the additional communications channel, 65 a method for performing a client request issued by the client to a first intermediary, the method comprising the steps of:

using the client to create the client request;

using the client to create an executable access control program, the access control program encoding a specification of a set of access rights to be delegated from the client to the first intermediary;

using the client to associate the access control program with the client request;

using the client to request an authentication ticket;

using the additional communications channel to transmit the request for the authentication ticket to the authentication server:

using the authentication server to issue an authentication ticket:

using the additional communications channel to transmit the authentication ticket to the client;

using the client and a communications channel from the client to the first intermediary to transmit the client request and its associated access control program from the client to the first intermediary;

using the first intermediary to generate a first intermediary request;

using the first intermediary and a communications channel to issue the first intermediary request and to transmit the access control program along with the first intermediary request thus issued;

using the server to receive a service request and the access control program;

using the server to make a determination whether the client approves the service request by performing a test that comprises the steps of

using the server to execute the access control program, and

using the server to check a value returned by the access control program thus executed; and

if and only if the determination thus made by the server is that the client approves the service request, using the server to execute the service request, and otherwise using the server to deny the service request.

28. The method of claim 27 wherein the additional communications channel is a secure channel.

29. The method of claim 27 wherein the step of using the server to make a determination whether the client approves the first intermediary request further comprises an additional test of using the server in conjunction with the authentication server and a channel between the server and the authentication server to verify that the access control program is authentic and is the client's.

30. The method of claim 29 wherein the channel between the server and the authentication server is a secure channel.

31. In a system comprising a client, a server, a number of intermediaries, the number being greater than or equal to one, a plurality of communications channels that connect the client, the server, and the intermediaries, and computing hardware to execute the client, server, and intermediaries and to support the communication channels, a method for performing a client request issued by the client to a first intermediary, the method comprising the steps of:

using the client to create the client request;

using the client to create an executable access control program, the access control program encoding a specification of a set of access rights to be delegated from the client to the first intermediary;

using the client to associate the access control program with the client request;

using the client and a communications channel from the client to the first intermediary to transmit the client request and its associated access control program from the client to the first intermediary;

using the first intermediary to generate a first intermediary 5 request;

using the first intermediary and a communications channel to issue the first intermediary request and to transmit the access control program along with the first intermediary request thus issued;

using the server to receive a service request and the access control program;

using the server to make a determination whether the client approves the service request by performing a test 15 that comprises the steps of

using the server to execute the access control program, and

using the server to check a value returned by the access control program thus executed;

if and only if the determination thus made by the server is that the client approves the service request, using the server to execute the service request, and otherwise using the server to deny the service request;

using the client to designate a revocation object at the <sup>25</sup> server associated with a right to be delegated that is encoded in the specification of the access control program:

if and only if the server executes the service request, using the server to test for the existence of the revocation object thus designated; and

if and only if the server finds that the revocation object exists, using the server to grant the delegated right associated with the revocation object, and otherwise using the server to deny the delegated right associated with the revocation object.

32. The method of claim 31 further comprising the step of using the client to revoke the right associated with the revocation object by eliminating the revocation object prior to the server's execution of the service request.

33. The method of claim 31 wherein the server is a file server and the revocation object is a file stored by the server.

34. The method of claim 31 wherein the client designates the revocation object independently of the server.

35. The method of claim 31 wherein the client designates the revocation object at run time contemporaneously with the step of creating the client request.

36. In a distributed file system comprising a plurality of nodes, each node comprising a processor and memory, a plurality of processes including at least one client, at least one file server, and at least one intermediary, each process executing on its own unique node, and a plurality of com-

munications channels that connect the processes to one another, a method for performing a client request issued by the client to the intermediary, the method comprising the steps of:

using the client to create the client request;

using the client to create an executable access control program, the access control program encoding a specification of a set of access rights to be delegated from the client to the intermediary;

using the client to digitally sign the access control program;

using the client to associate the access control program thus digitally signed with the client request;

using the client and a communications channel from the client to the intermediary to transmit the client request and its associated access control program from the client to the intermediary;

using the intermediary to generate an intermediary request:

using the intermediary and a communications channel to issue the intermediary request and to transmit the access control program along with the intermediary request thus issued;

using the server to receive a service request and the access control program;

using the server to make a determination whether the access control program is valid by performing a test that comprises the steps of:

using the server to verify that the access control program bears a digital signature that is authentic and that is the client's;

using the server to verify the identity of the first intermediary; and

using the server to verify that the client has the rights that it purports to delegate via the access control program:

if and only if the server thus determines that the access control program is valid, using the server to make a determination whether the client approves the service request by performing a test that comprises the steps of: using the server to execute the access control program; and

using the server to check a value returned by the access control program thus executed; and

if and only if the determination thus made by the server is that the client approves the service request, using the server to execute the service request, and otherwise using the server to deny the service request.